ARRL AMATEUR RADIO

5TH COMPUTER NETWORKING CONFERENCE

ORLANDO FLORIDA — MARCH 9, 1986

Fifth ARRL Amateur Radio Computer Networking Conference

Coordinators:

Paul L. Rinaldo, W4RI

Gwyn Reedy, W1BEL

American Radio Relay League

Newington, CT USA 06111

ISBN: **0-87259-033-X**

First Edition

FOREWORD

This is a collection of technical papers presented at the Fifth ARRL Amateur Radio Computer Networking Conference in Orlando, Florida, March 9, 1986.

Since the Fourth Conference held March 30, 1985, packet radio has grown from approximately 4000 stations to over 11,000. And, the packet revolution is taking root worldwide. In November, 1985, the International Amateur Radio Union Region 3 Conference adopted the AX.25 link-layer protocol as an interim standard. Radio amateurs in Japan and England are hard at work on new satellite systems that will support packet communications.

In the past year, the problem has shifted from one of how to stimulate interest to one of how to ease congestion on 2-meter packet frequencies. Some of the cures are treated in these papers: moving to higher frequencies, higher speeds and fine-tuning of link-layer protocols. Also evidenced in these papers is intensified interest in network-- and transport-layer protocols, all of which contribute toward the development of new standards.

All papers contained in this publication were submitted in camera-ready form, are unedited and are solely the responsibility of the authors,

David Sumner, K1ZZ
Executive Vice President

Newington, Connecticut
March 1986

Trademarks appearing in these papers are:

AMSAT is a registered trademark of The Radio Amateur Satellite Corporation.
ANSI is a registered trademark of the American National Standards Institute
CP/M is a registered trademark of Digital Research Inc.
Heathkit is a registered trademark of the Heath Company.
IBM is a registered trademark of International Business Machines, Inc.
TELENET is a registered trademark of Telenet Communication Corporation.
TYMNET is a registered trademark of Tymshare.
UNINET is a trade name of US Telecom Data Communications Co.
Xerox is a trademark of Xerox Corporation.
Z80 is a registered trademark of Zilog, Inc.

CONTENTS

RF, RF, WHERE IS MY HIGH SPEED RF?

Terry Fox, WB4JFI
President, AMRAD
1819 Anderson Rd.
Falls Church, VA 22043

Abstract

This paper presents some thoughts of where we are now and where we are heading in the RF hardware evolution of Amateur packet radio. It may be a disappointment to some in that it raises more questions than it answers. I feel it is vital ot raise these questions now, since this is where we need the most work.

Amateur Radio is a hobby for 'Radio' operators. It seems that a lot of us (myself included) have either forgotten that, or aren't willing to spend the time in RF design any more. Even when we try to work on RF problems (such as channel congestion) we tend to try clever 'digital tricks" rather than look at the basic problem. Boy have we become spoiled! While we have advanced greatly in a short time in the digital end of packet radio, our RF technology lags further and further behind (Just by staying at the same place). There have been only one or two contributions made in the last year or so.

This is particular1y bad when one looks at the overall picture of the Amateur Packet Radio Network. For the last six months all I have heard is the same question: 'Where is the Network Layer?" I hate to be the bearer of bad news, but that isn't where our efforts should be concentrated. The Network Layer camps are progressing fine, which just adds to the problem. How can this be? The answer is in the term "'overhead". It doesn't matter if Virtual Circuits or Datagrams are used, both will add a lot of data that must be passed over the same already crowded RF paths. This means either more packets or longer packets will be flying through our RF. If you think our channels are congested now, just wait until the packet switches come on-line!

My answer to the question of where do we need the most work in packet radio right now is therefore simple, ALL PHASES OF THE RF WE USE! Having said that, I must now back down and admit that I cannot/will not alter my own course and proceed into the RF domain, I am too involved in the Network Layer development. The above comments are meant as a challenge to all those who do have the expertise to work in RF. WE DO NEED YOUR HELP! We need newer and faster radio and modem designs, NOW.

HE Packet Operation

In the last year there has been a large growth in the amount of HF packet operation. Almost all of this operation has been centered around the frequency of 14.103 mHz. The present technology being used is 200 Hz shift AFSK 300 bps lower sideband. Since this one frequency is being used both in the U.S. and Europe, it has become quite crowded most of the time. To add to the problem, some complaints have been surfacing about interference from the packet operation to nearby DX beacons operating on 14.100 mHz, which are used to detect band openings.

Bob Bruninga, WB4APR has been operating an HF station/gateway on 10.147 (USB) for over a year now. As more equipment is becoming available to operate 30 meters, it might be time to move some 0f the auto-forwarding stations from 20 meters-to 30 meters. This might help reduce the congestion on 14.103, show more activity on 30 meters, and help the DX amateurs all at the same time. I would hope to see more operation on 10.147 mHz over the next year.

As far as technological improvements on HF goes, I haven't heard of much lately. Paul Rinaldo still has a couple of Packet Adaptive Modem (PAM) prototypes built, but no one has done any serious experiments with them. These modem devices were described in the Second ARRL Cornputer Networking Conference Proceedings and are a first step toward minimum-shift-keying (MSK) operat ion on HF. We need someone to pick up the ball and test these devices, or alternatively come up with some other scheme of increasing the data speed on HF packet.

VHF/UHF Packet Operation

Amateur packet radio is really taking off on VHF and UHF. Some of the common bands and frequencies of use are as follows:

Two Meters (These are fairly standard nation-wide):

145.010 mHz (mostly National Backbone).
145.030 mHz (some local net operations).
145.050 mHz (mostly local net operation)
145.070 mHz (some local net operation).
145.090 mHz (some local net operation).

220 mHz, Low speed channels (1200 bps):
(these have been requested from T-Marc in the D.C. area, the last five may have other services on them by now).

221.010 mHz (some east-coast backbone).
221.030 mHz
221.050 mHz
221.070 mHz
221.090 mHz
221.110 mHz
221.130 mHz
221.150 mHz
221.170 mHz
221.190 mHz

220 mHz wide-band channels (100 kHz channels centered around the following freq):
(again specifically requested of T-Marc in D.C.)

220.550 mHz
220.650 mHz
220.750 mHz
220.850 mHz
220.950 mHz

440 mHz narrow-band channel (1200 bps):
(again, requested of T-Marc in the D.C. area>

441.000 mHz

440 mHz wide-band channels (100 kHz bandwidth centered around the following freq.):
(requested of T-Marc in the D.C. area)

430.050 mHz
430.150 mHz
430.250 mHz
430.350 mHz
430.450 mHz
430.550 mHz
430.650 mHz
430.850 mHz
430.950 mHz

I should point out that the above frequencies have NOT been set aside specifically for packet radio at this point, but rather may be available for use IF WE NEED THEM.

They were requested to be set aside for packet use by one of the local Washington D.C. clubs (Tri-State Amateur Radio Club). We need to get some radios up on these frequencies before yet another voice repeater uses them up, or they are given to some other spectrum-starved service.

At this time there has not been much work in trying to get frequencies assigned to packet radio in either the 900 mHz band or the 1215 mHz band (in the Washington D.C. area at least).

Michigan Packet Radio Frequency Plan

Another packet radio frequency plan surfaced recently from Michigan. Apparently there was a state-wide meeting in mid-November of 1985, the results of which are as follows:

144-148 mHz

Apparently the Michigan Repeater Council does not coordinate packet channels on two meters, so the following is by "gentleman's agreement" rather than an official coordination:

144.910 mHz (Experimantal and QRP)
144.930 mHz (Local Area Network)
144.950 mHz (Local Area Network)
144.970 mHz (Local Area Network)
144.990 mHz (Non-Digipeated simplex)
145.010 mHz (Inter-Lan & Forwarding)
145.030 mHz (Local Area Network)
145.050 mHz (Local Area Network)
1451070 mHz (Local Area Network)
145.090 mHz (Experimental and QRP)

The Michigan group took a different approach for 220 mHz. They reserved several frequencies for full-duplex low-speed repeaters and simplex digipeaters, and four freqs for 9600 bps test channels, but pushed the higher speed packet operation up to 430 mHz, with only 50 kHz channels even there. The rest of their bandplan is as follows:

220 mHz low-speed (1200bps) full-duplex repeater channels:

OUT	IN	OUT	IN
220.52 mHz/	222.12 mHz	220.64 mHz	222.24 mHz
220.54 mHz/	222.14 mHz	220.66 mHz	222.26 mHz
220.56 mHz/	222.16 mHz	220.68 mHz	222.28 mHz
220.58 mHz/	222.18 mHz	220.70 mHz	222.30 mHz
220.60 mHz/	222.20 mHz	220.72 mHz	222.32 mHz
220.62 mHz/	222.22 mHz		

220 mHz simplex low-speed channels:

220.74 mHz, 220.76 mHz, and 220.78 mHz

220 mHz 9600 bps channels (uncoordinated):

220.825 mHz, 220.875 mHz, 220.925 mHz, 220.975 mHz

There would be nineteen coordinated 50 kHz channels for linking in the 430-431 mHz band segment as follows:

430.025 mHz	430.275 mHz	430.525 mHz	430.775 mHz
430.075 mHz	430.325 mHz	430.575 mHz	430.825 mHz
430.125 mHz	430.375 mHz	430.625 mHz	430.875 mHz
430.175 mHz	430.425 mHz	430.675 mHz	430.925 mHz
430.225 mHz	430.475 mHz	430.725 mHz	

The above information is being given here primarily to show that there are frequencies out there (for most of the country at least), and that the packet community IS being recognized and served by the frequency coordinating bodies. It is also being given to (hopefully) aid in the development of RF equipment for these bands.

Simplex vs Full-Duplex Digipeater Operation

Way back when, when packet radio was still mainly in Vancouver, I was pushing the use of full-duplex repeaters. At the time I was convinced of the error of my ways (at K8MMO's house one night, I remember). A simplex digipeater is MUCH easier and cheaper to put up. With just a TNC and a radio you too can put up a simplex digipeater. The only disadvantage of a simplex digipeater is the lower throughput. Is the loss of throughput made up by the cheaper cost of simplex digipeaters? Well, I'm not sure, but I feel it is time once again to look at repeater operation.

Some of the advantages of simplex digipeaters are:

A. The cost of a simplex digipeater is MUCH smaller than that of a full-duplex one.

B. The complexity of a simplex digipeater is also much less, especially in the RF plumbing required (ie. filters).

C. A simplex digipeater is also physically smaller, allowing it to be placed in remote areas more easily.

D. Since there is less equipment, it is less likely to require maintenance, and less control circuitry may be needed.

E. In order to use full-duplex to full advantage, user radios should also be capable of full duplex operation, driving the user cost up drastically.

The only real disadvantage to simplex digipeaters is that it has less throughput than a full-duplex type repeater. Some of the reasons for this reduced throughput follow.

One reason often mentioned for reduced throughput of a simplex digipeater is that it time-shares the same frequency for both receive and transmit, thus reducing the throughput by at least 50%. Actually, this is not completely true when one considers that full-duplex operation uses two frequencies. If two separate simplex digipeaters were put on the frequencies used by a full-duplex digipeaters, almost all of the channel capability can be recovered. The only amount of channel capability still lost would be the receive/transmit turn-around time, both at the individual stations and at the digipeater.

The other major loss of channel throughput in a simplex digipeater system is due to the hidden station syndrome. Since stations using the same digipeater may not be able to hear each other, one station may start transmitting a packet to the digipeater while another station was already sending a packet to the digipeater, or some other station on the same frequency, causing a collision and possible loss of data. In full-duplex digipeater operation this would be much less likely to happen, since all other stations would hear any station that starts to transmit on the repeater input (except for the small transition time between receive and transmit at the individual stations) on the repeater output.

Cross-Band Operation

Some hams have suggested that in order to reduce the amount of plumbing needed at digipeaters that the digipeaters receive on one band and transmit on another band. This cross-band operation would allow full-duplex operation at reduced cost and size.

Channel Access Methods

How stations gain access to the RF channel to pass data is another descision that must be made. Some of the different systems to chose from are listed below.

Aloha Type Channel Access Method

The first major packet radio network was the Aloha Network built in Hawaii. The Aloha Network used the RF channel by having a station immediately transmit data whenever it had some to send. Collisions of transmissions were detected by not receiving an acknowledgement of the data by a certain time. The theoretical maximum channel utilitization using pure Aloha is 18%.

Slotted Aloha Channel Access Method

One of the prolblems with pure Aloha is that anyone can transmit packets at any time. One method to increase channel utilitization is to divide the potential transmit time into discrete time-slices or slots, each of which is slightly longer than the time it takes to send a packet.

Once the stations are synchronized (usually by having a master station transmit a short clock pulse), a station will only transmit at the beginning of a slot. This reduces the amount of collisions, since stations will no longer accidentally transmit part-way through another stations packet transmission. Packets either make it through fully, or are fully destroyed. Using Slotted Aloha just about doubles the channel throughput to about 37%.

Reservation Aloha Channel Access Method

Another method used to improve channel utilitization over pure Aloha is to reserve specific time slots for each station to transmit data. There are several different schemes as to how these reservations are made and maintained, but basically they all assign times for stations to transmit, thereby greatly reducing collisions.

Token-Type Channel Access Method

Yet another method of controlling access to the data channel is to allow transmissions by a station only when it has "permission". This permission is usually in the form of a "token". This token is passed back and forth by all the stations on the channel. When a station receives the token, it checks to see if it has data to send. If it does, it sends the data, then passes on the token to the next station. If it has no data, it immediately passes on the token to the next station.

Among the disadvantages to token type access methods is that they must be carefully supervised. In order for additional stations to be accepted onto the channel, they must somehow be added to the token-passing list. The easiest way to have this function properly is to have a master station monitor the token passing and allow new station(s) in whenever it has the token.

Carrier-Sense Multiple Access (CSMA)

The method we Amateurs presently use to gain access to the RF channel is through a system called Carrier-Sense Multiple Access, or CSMA. CSMA is a fancy term for how we hams have shared our spectrum space all these years. Basically it means you are supposed to listen for others before you transmit. If you hear someone else on the channel, you wait until they are done before you start transmitting. With many people competing for the same channel, CSMA is a good method to control channel access.

The biggest problem associated with simple CSMA is what is called the "hidden transmitter" situation. There is a possibility that whenever a half-duplex channel is used, not all stations can hear all other stations. In fact, with the Amateur Packet Network of today with simplex digipeaters, this is quite likely. Whenever a station is hidden from another station, the possibility of both of them transmitting at the same time exists, since they cannot sense each other. The possibility of collisions expands greatly with the addition of each station that cannot hear another station. The degradation of the channel quickly reaches the point where there are more collisions than normal transmissions. Several schemes have been devised to overcome this situation.

One system for recovering from collisions in a CSMA enviroment involves "persistence". Persistence has to do with how a station handles access to a busy channel when it has data to send. Suppose a station has data to send, but detects the channel is busy. One method of handling this situation is for the station to transmit its data as soon as it thinks the channel is free. This is called 1-persistence because the probability of the station transmitting when detecting the idle channel is 1.

The other extreme in persistence is called nonpersistent CSMA. In this case, when the station detects the busy channel, it doesn't wait for the channel to become free and immediately transmit. Rather, it waits a random amount of time and then tests the channel again. If the channel is free, it will transmit. If the channel is still busy, the station repeats the wait and test cycle until the channel becomes free.

A third persistence scheme is called p-persistent CSMA. This is a compromise between the above two systems. When a station has data to send, it will first check for an idle channel condition. If the channel is free, it will transmit its data with a probability of p. If the channel is busy, the station will wait until the channel is idle before invoking the probability test for transmission. If the value of p is different for each station, this can reduce the potential for collisions of transmissions.

CSMA with Collision Detect

A modification to the basic CSMA operation involves monitoring the channel while transmitting. This means a full-duplex channel is being used. If, while transmitting data, a station detects that its data is not what's on the channel, another station has taken over the channel and the monitoring station should stop transmitting. Since all stations can now hear all other stations, the hidden station problem is almost completely eliminated. This collision detection can help reduce the number of collisions, at the cost of all stations being required to operate in full-duplex mode. If full-duplex capability is possible, CSMA-CD can be the most effective use of a channel.

Busy-Tone CSMA System

Another method of reducing collisions on a CSMA channel is to use a busy-tone to indicate when the channel is being used. This requires that the channel is being controlled by a master station with which the individual stations communicate. It also reguires that a secondary channel be available to transmit a busy signal. This secondary channel does not have to be a full data type channel as the presence or absence of a busy signal is the only information carried on the secondary channel. Whenever the master station is receiving data from a station, it sends out the busy-signal on the secondary channel. Once the main data channel is free, the master drops the busy-signal. The individual stations monitor the channel for the busy-signal before transmitting data. This system also helps reduce the amount of collisions due to hidden stations, although not as effectively as the collision detect mechanism described above.

While this busy-signal system does improve channel access, it is not as effective as the CSMA-CD system mentioned above, for about the same complexity. This busy-signal system has been tried on the Amateur bands, and it is more effective than the simple CSMA we presently use.

With the above in mind, lets look at how these trade-offs are made in two different parts of the packet radio network.

Individual User/Local Network Operation

We are using simplex digipeaters almost exclusively at this point. I believe this is the best use of the RF channels we use at this point. Full-duplex repeater operation is not feasible on two meters with our present allocation, since the five main channels are right next to each other In frequency. In some areas, full-duplex (voice type) repeater assignments may be available, but are the users willing to pay the added costs to put up a full-duplex digipeaters? From what I have seen, the answer is no. It appears that a more palatable solution to channel congestion is to put up more digipeaters. Brian, WB6RQN tried to run an experiment by putting his Unix system on the AMRAD voice repeater (147.81/21 mHz). This test was met with a great yawn. Admittedly, the repeater lost some coverage part of the way into the experiment due to loss of antenna height at the repeater. I don't think this was the main reason for the lack of interest. I think that most everybody has accepted simplex digipeater operation for packet use for now.

Cross-band operation is not a viable option for the user/Local net due to its added expense for a second rig and antenna system at every location.

I mentioned that some hams have used a busy-signal system with digipeaters to reduce

collisions. If the users are willing to pay for another receiver (or be lucky enough to find a repeater pair to use), this system may be a viable alternative. It does greatly reduce the hidden station problem, without requiring full-duplex operation at all locations, as CSMA-CD would.

My conclusion however, is that at this time simplex repeater operation is still the best way to go for Local Network operation. In the future this may change, especially if the Local Network becomes more of a cellular type operation at 900 mHz. For now, if a channel get too congested, it is easy enough to put up another simplex digipeater (ala FM voice repeater growth).

Local Network vs Backbone Network

I wanted to mention briefly that there should be a re-thinking of how our packet channels are being used. Both the Michigan Bandplan and the Tri-State frequency requests indicate the use of some frequencies for "local networks", while other frequencies have been set aside for "network backbone" operation. This is a very important point. We should start using the "local network" frequencies for small areas, such as small towns, or one part of a metropolitan area. These local network frequencies would be re-used by other small groups far enough away to avoid complete RF overlap (if we use simplex digipeaters, the occasional overlap won't affect operation much). The local networks would then have access to the backbone via dual-port digipeaters or multi-port packet switches. A local area digipeater or packet switch should only cover as much as it needs to for the local group. This frequency re-use works almost like the cellular system, where each local network is a "cell" and the cells are hooked together by the backbone.

The main idea I want to get across is that putting up a super-digipeater for a "local" network can be counter-productive. The super-digipeaters work best for the backbone, where they need to reach as far as possible (remember, we can only chain eight of them together). Keep the "local network" digipeater coverage within the "local" area.

Network Backbone RF Operation

There have been many different suggestions for how we should construct the RF part of the network backbone. Presently, we are using simplex digipeaters here also. Once again, I think this is mainly because they are cheap to put up. Another reason is that up until recently, if a digipeater was to use something other than the main frequency, it would it have to use two TNCs and two radios.

One of the common suggestions made for use of our frequencies for the network backbone is to imply direction by frequency. For example, if a packet is to be routed south from a switch, the packet would be sent on one frequency. Packets going north would be sent on a different frequency. East and West bound packets would be sent on still different frequncies.

This idea requires a lot of smarts in the switch, so it can control which radio channel to send the data, and therefore which radio. We should be able to do this someday, but it may be beyond our reach in the near future. In addition, this adds a lot to the cost, size, and antenna requirements of the digipeater or switch.

What's Happening Now?

Present packet operation is still mostly on two meters at 1200 bps using Bell 202 type modems. As an example, almost all of the east-coast backbone operation is still on 145.010 mHz.

One improvement is that some local areas have begun using the other two meter packet channels as local networks as suggested above. These local networks are usually concentrating around a local packet bulletin board (PBBS). In the Washington D.C. area for example there are at least two different groups using 145.050 mHz for local network traffic. The use of "Super-Digipeaters" seems to be fading, except for the backbone. In order to reduce the channel congestion (especially while using half-duplex digipeaters) this trend to localized networking will be very beneficial.

At the Fourth ARRL Computer Network Conference (March 30, 1985 in San Francisco) Steve Goode, K9NG described a method of modifying the Hamtronics FM-5 220 mHz tranceiver for 9600 bps operation. Since then TAPR has made boards available for this modification, and some hams around the country have tried this mod on various rigs, with different degrees of success. Unfortunately, I have heard some negative comments about the use of these "off-the-shelf" rigs when modified for high-speed packet operation, especially in uncontrolled enviroments (such as one might expect at a digipeater like WB4JFI-5).

One suggested method of curing some of the problems encountered is to replace the standard IF filter(s) in the modified radios with slightly wider ones, allowing more "slop" in the IF bandwidth.

Moving to 220 mHz, The First Step

In order to reduce the amount of traffic on the network backbone, I propose that we make the first step to 220 mHz operation. This step is fairly simple, we just add a 220 mHz 1200 bps digipeater wherever there is a two meter digipeater in the backbone. This parallel path can be enchanced at various points by installing a dual-port digipeater (ala Jon Bloom, KE3Z and a Xerox 820 board) instead of the normal TNC type digipeater. This parallel path will allow us to check the RF paths and also provide more data throughput on the network. Heavy users of the backbone (PBBS's) could then use one path while the other path could be used for lighter traffic. Another advantage of going to 220 mHz even at 1200 bps is that it reduces the number of stations that have access to the backbone directly, thereby reducing channel occupancy, since not as many hams have 220 mHz capability.

AMRAD should be making this first step about the time of this conference. We are improving WB4JFI-5 to a dual-port digipeater with access on both 145.010 mHz and 221.010 mHz, both at 1200 bps for now.

9600 bps on 220 mHz

As mentioned earlier, some effort has been made to design higher speed radios at 220 mHz. Steve Goode, K9NG did a good job on his "modem" modification to the Hamtronics FM-5. Even so, some hams have found that they have to continually fight the rig to keep it operating properly. I feel that this is at least partially due to the fact that the FM-5 is still basically a voice type radio, optimized for voice operation (especially in the bandwidth department). What we really need is a radio that was designed from the ground-up as an "RF-Modem" rather than a voice rig.

Another person that has felt that way is Gary Fields of Boston, Mass. He has been working for a while now on a complete 220 mHz radio that is designed to be an RF modem. While he hasn't finished it yet (last I heard) it sounds promising.

Another effort being made on the 9600 bps 220 mHz front is being done by the AMRAD crowd. Chuck Phillips, N4EZV (of spread spectrum fame) and Andre Kestloot, N4ICK are working on a 220 mHz RF-Modem design. It is suprising how much like a modem it looks. They are presenting a paper on their design ideas elsewhere in these proceeding so I will not steal their thunder here.

I hope 1986 will be the year for 9600 bps 220 mHz packet radio, its overdue and needed desperately.

56kbps Design Request by ARRL Digital Committee

The ARRL Digital Committee met last December at Newington, CT and one of the items on the agenda was (suprise!) higher speed radios. The committee came up with a wish list for the design of a digital radio for high-speed data. The basic design involves the use of a data interface/IF modem followed by a transverter to the band of choice. The IF frequency in/out of the data/IF interface should be 28 mHz with RF levels to match

standard transverters (around 10 mw?). The data interface should accept standard serial data signals at either TTL or RS-422 levels. The data signals should be as follows:

Signal	Data/IF	TNC
RxData	--->	
TxData	<---	
RxClock (X1 speed)	--->	
TxClock (X1 speed)	--->	
Data Carrier Detect	--->	
Request-to-Send	<---	

The signalling speed of the complete data/IF/transverter chain should be at least 56 kbps, preferably using standard FSK FM modulation. The bands of operation should include 220 mHz, 440 mHz, and possibly 900 and 1215 mHz. It should operate in a 100 kHz channel, and should provide a clean RF output. Full duplex operation of the Data/IF interface should be possible, and the Rx/Tx switching speed should be extremely fast.

Anyone wishing a nice RF challenge should look no further. A radio that meets the above specs would be very welcome indeed!

Conclusion

Unfortunately, we have not progressed in the RF portion of packet radio development as quickly as we need to. There is still a lot of room to experiment with radio designs. The should be a welcome challenge to some enthusiastic Amateur out there who knows RF, and wants to learn about digital transmission methods and packet radio.

There is a world of variations available for the RF digital designer. Just tell us what you need, and we will supply the bits! If this sounds a bit like I am begging, I am. I WANT FASTER RADIOS!!

References

Rinaldo, P. L., "Introducing: the Packet Adaptive Modem (PAM)", Second ARRL Amateur Radio Computer Networking Conference, ARRL, 1983

Brooker, J., et al, "Michigan Packet Radio Frequency Plan", Packet Radio in Southern Michigan (PRISM), Nov. 17, 1985

Tanenbaum, A. S., "Computer Networks", Prentice Hall, 1981

Goode, S. "Modifying the Hamtronics 'FM-5 for 9600 bps Packet Operation"', Fourth ARRL Amateur Radio Computer Networking Conference, ARRL, 1985

Phillips, C. and Kesteloot, A. "A High Speed R.F Modem", Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986

Kesteloot, A., "An Application Note Describing A High-Speed Phase-Coherent FSK Generator", Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986

A HIGH-SPEED RF MODEM

Chuck Phillips N4EZV
Andre Kesteloot N4ICK
AMRAD
P.O.Drawer 6148
McLean VA 22106-6148.

Introduction:

Although the present 1200 baud Bell 202 standard is perfectly acceptable for normal packet-radio QSOs, the amateur radio community needs to be able to communicate at higher baud-rates if an inter-city trunk line, or backbone network, is ever to be implemented.

Many believe that the next generation of modems should operate at a minimum of 9,600 bauds. Steve Goode K9NG was able to successfully modify the Hamtronics FM-5 to operate at 9.6KBd (1) while another approach, that of Gary Field WA1GRC, is mentioned in the 1986 edition of the ARRL Handbook (page 19.37). From what the authors have seen of Gary's design (2), his approach appears very promising.

Motorola too has long been involved in the field of RF modems (see for instance ref.3) and has recently filed a petition with the FCC to allow for the creation of "radio frequency local area networks"(4). (The above list is by no means exhaustive and simply represents some of the designs with which the authors are familiar.)

This paper will describe the approach followed by several members of the Amateur Radio Research and Development Corporation (AMRAD) to design a high-speed RF modem. Another paper, presented to this Conference in the form of an Application Note, describes the actual circuitry used to generate phase-coherent FSK.

Guidelines:

This modem is intended for the radio amateur community and as such, from the start of the project, the design criteria have included:

- ease of duplication,
- use of readily available parts and
- minimum of adjustments to be carried out with only simple test equipment.

It is intended that the final design will operate at up to 56Kbd on 220MHz and 440MHz. This paper describes an interim version already operating in a breadboard configuration.

The RF Modulator:

Fig.1 shows the method employed to generate phase-coherent FSK. (The use of two frequencies not phase-related would introduce unacceptable splatter.) A 2.2MHz crystal-controlled master oscillator drives a dual-modulus counter. The RS-232 level data input (TXD) is translated to TTL and used to change the dividing ratio of the counter and thus generates two phase-related discrete frequencies, in our example 200KHz and 220KHz. This is in fact equivalent to having a 210KHz carrier shifting + and - 10KHz. (Should an 40KHz total shift be desired for higher baud rates one could use a 4.4MHz crystal in a similar set-up. The result would be a 420KHz carrier shifted + and -20KHz.)

A gate, when enabled by the RTS line, then feeds the output of the dual-modulus divider to a doubly balanced mixer. The output of the mixer is a 10.7MHz IF which is then heterodyned with a 210MHz local oscillator in another doubly balanced mixer to produce an RF signal in the 220MHz band.

The Demodulator:

Figure 2 shows the receiver portion, consisting of a low-noise 220MHz stage followed by a doubly balanced mixer where the local oscillator frequency (210MHz) is injected. The resulting IF output at 10.7MHz drives, via a trifilar- wound transformer (Mini-Circuits PSC2), two identical doubly balanced mixers (MC1496). Both have ceramic resonators output circuits tuned to 455KHz, but one has a local oscillator injection at 10.255KHz whereas the other has a L.O. of 10.235KHz. Thus when the IF output is 10.710MHz (mark), the doubly balanced mixer with the 10.255KHz L.O. produces an output at 455KHz and conversely, when the IF swings to 10.690 MHz (space), it is the other doubly balanced mixer which produces a 455KHz output. In the scheme just described, there are 8 major heterodyne products:

10,710KHz	-	10,255KHz	= 455KHz
10,690	-	10,235	= 455
10,710	-	10,235	= 475
10,690	-	10,255	= 435
10,710	+	10,255	= 20,965
10,690	+	10,255	= 20,945
10,710	+	10,235	= 20,945
10,690	+	10,235	= 20,925

Note that only the first two combinations can produce an output from one of the two 455KHz stages. The other heterodyne products fall outside the 455KHz bandpass and are eliminated.

The two outputs are then detected and fed to a comparator,the output of which is RXD. It is to be noted that no tuning is needed in the process.

The Synthesizer:

For the sake of clarity, figures 1 and 2 have shown the local oscillators as being crystal controlled. In fact, the unit uses a master crystal oscillator (at 2.2MHz in this particular case) whilst all other oscillators are synthesized. The 2.2MHz crystal oscillator output (located in the RF modulator section, see fig.1) is fed to a divider by 440 to produce a 5KHz reference frequency. Presuming the stability of the crystal to be +/- 0.001% (easily achievable with an oven) the crystal oscillator output can drift from 2,199,978Hz to 2,200,022Hz. At the output of the divider by 440, the 5KHz reference will thus drift between 4,999.95Hz and 5,000.05Hz, equivalent to a maximum drift of 0.1Hz.

All the local oscillators are designed along the following lines and thus only one will be described: the output of a 10.490MHz free-running voltage-controlled oscillator (VCO) is fed to an MC145151 divider-by-2,098 stage, thus producing approximately 5Khz. This output is compared to our reference 5KHz source in a phase comparator, the output of which is a variable DC voltage applied to the 10.490MHz VCO as negative feedback. Note that once phase-lock has been achieved, the output of the divide-by-2098 divider will drift in phase with, but no more than the 5KHz reference oscillator. Using the maximum drift calculated above, the 10.490MHz oscillator will only be able to drift 2098 x 0.1 Hz, or +/- 100Hz. All other oscillators phase-locked to this 5KHz reference will exhibit essentially the same stability.

An added advantage of this scheme is that the output of the RF modulator can be set to any discrete channel on 220MHz, and so, of course, can the receiver. It is thus possible to contemplate offsets for duplex operation, etc. Since the dividing ratio of each phase-locked loop is determined by a set of DIP-switches (the MC145151 can be preset to divide by any integer between 3 and

16383), changing the configuration of the equipment is an extremely easy task.

Conclusion:

The scheme described above is simple, easy to duplicate and only an oscilloscope and a frequency counter are needed to adjust the modem. It is anticipated that the major parts needed for this modem (including a printed circuit board) will be rnade available through a distributor.

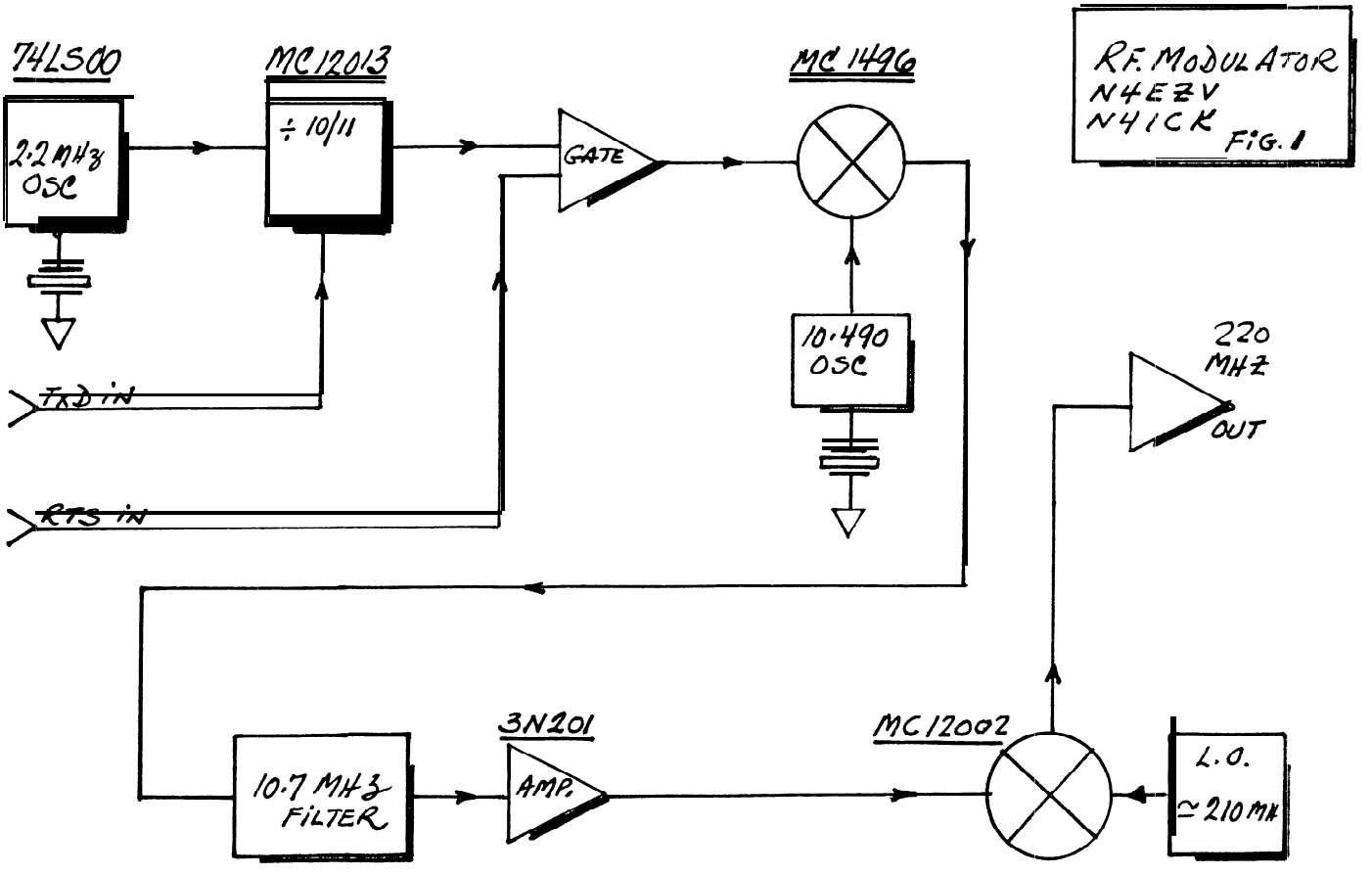
References:

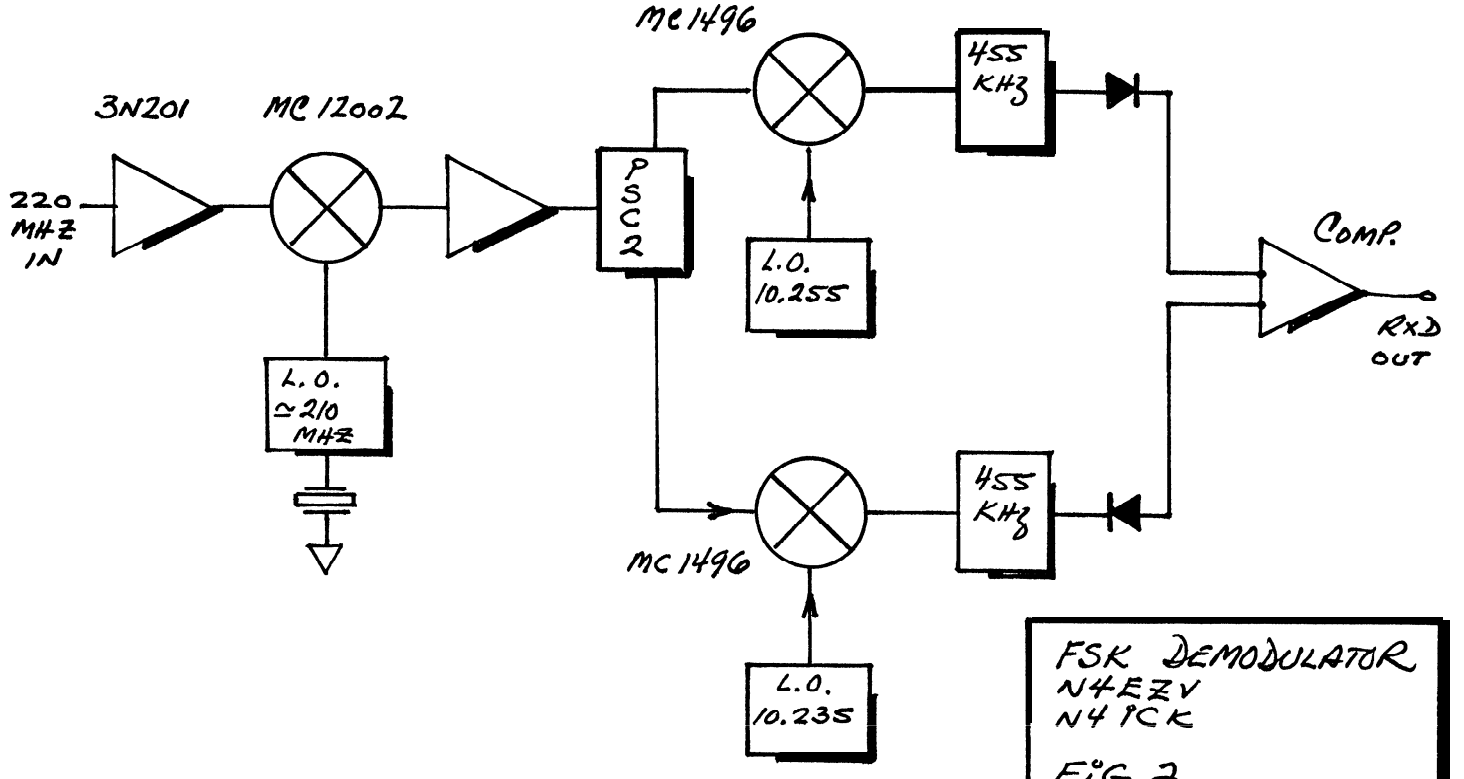
(1) GOODE Steve K9NG, "Modifying the Hamtronics FM-5 for 9600 bps Packet Operation." ARRL Computer Networking Conferences 1-4, pp 4.45-4.1

(2) Personal communication dated September 28, 1985.

(3) HATCHETT John and HOWELL Bill Motorola Semiconductors, Phoenix, AZ. "RF Modems", RF Design Magazine, Sept/Oct and Nov/Dec 1984.

(4) "Motorola Proposes Wireless Lan" Personal Communication Technology Magazine, Nov 1985 p.38





FSK DEMODULATOR
N4EZV
N4PCK

FIG. 2.

AN APPLICATION NOTE DESCRIBING A HIGH-SPEED PHASE-COHERENT FSK GENERATOR

Andre Kesteloot N4ICK
AMRAD
P.O.Drawer 6148
McLean, VA 22106-6148

Introduction:

This application note is a companion paper to the one entitled "A High-Speed RF Modem" also being presented to this conference. It describes a phase-coherent FSK generator built entirely with readily-available components and requiring only a frequency counter for set-up.

Circuit Description:

Figure 1 shows the circuitry used for our particular application. One quarter of a 74LS00 is configured as a 2.2 MHz crystal oscillator, driving two buffers, one between the oscillator and a divide-by-440 MC145151 stage having a 5KHz output (to drive phase-locked loops used elsewhere in the RF modem) and the other feeding an MC12013 dual-modulus divider chip.

This chip divides its input signal by 11 when its pin 9 is brought low, and divides by 10 when than pin is held high. The output at pin 2 is thus either 200KHz or 220KHz, depending only on the voltage level at pin 9. This output is amplified by a 2N2222 stage and then applied to the last quarter of the 74LS00, configured as an RTS gate. When RTS is high, the gate is enabled, and the 200KHz (or 220KHz) signal is applied to one of the inputs of a MC1496 doubly balanced mixer. The other input is fed with a 10.490MHz signal (shown on this schematic as a crystal oscillator for the sake of clarity, but in the actual modem generated by a frequency synthesizer.)

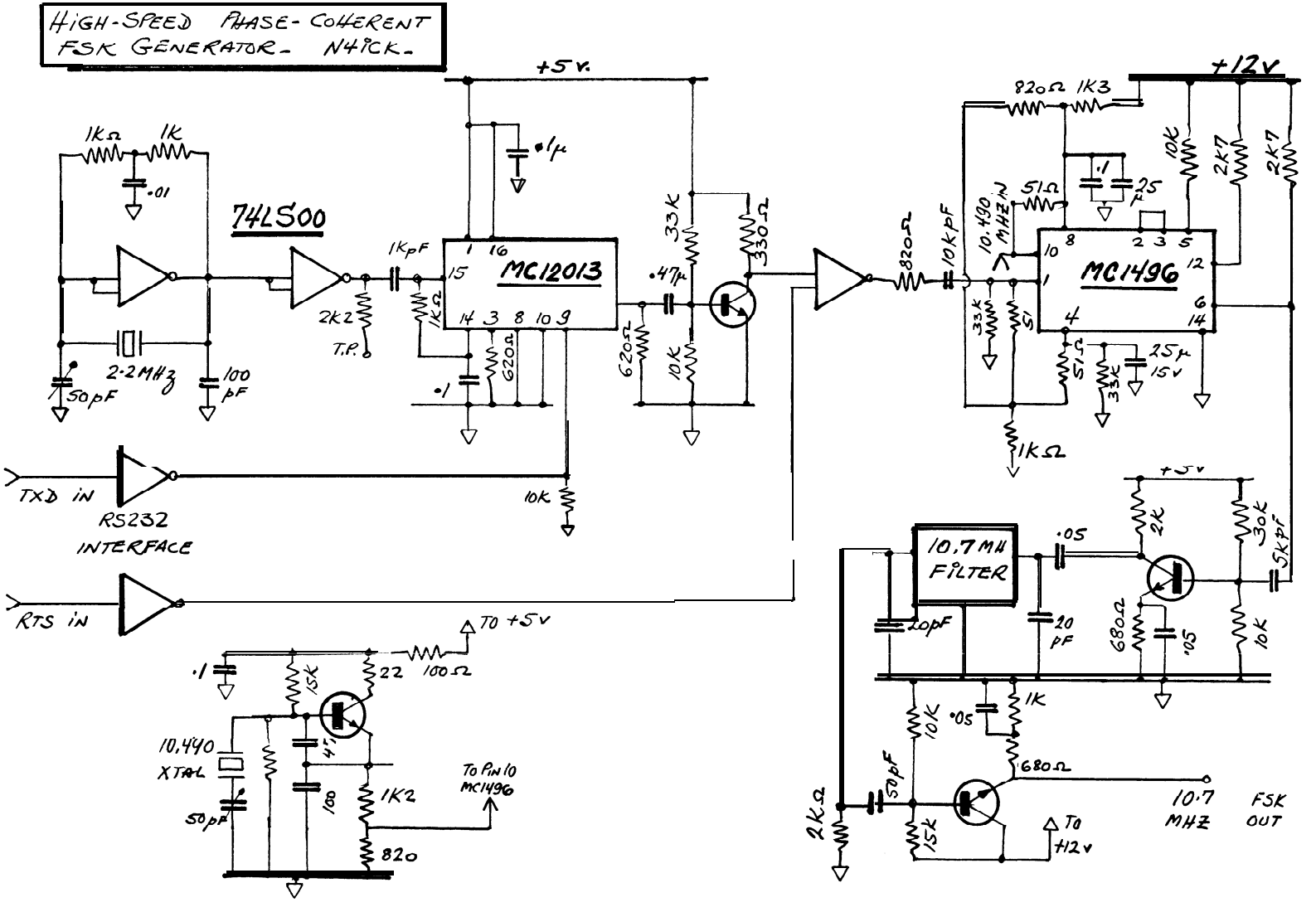
The output of the MC1496 is amplified by a single 2N4401 stage and applied to a 10.7MHz KVG crystal filter with a 40KHz bandwidth, The output of the filter is buffered by means of a 2N2222 emitter follower stage.

Motorola uses a MC12013 in an RF Modem with a data rate of 1.5Megabits (!), and we should thus have no fear that our packet system will reach the upper limits of that chip in the near future!

Should a total frequency shift of only 10KHz be desired, a divider-by-2 stage should be inserted between the 2.2MHz oscillator and the MC12013 dual-modulus stage. The output would now be 100KHz and 110KHz, and if the same type of 10.7MHz output filter is to be used, the crystal for the oscillator will have to be cut for operation at 10.595MHz. Similarly, a 40KHz total frequency shift can be obtained by using a 4.4MHz reference oscillator and a 10.280MHz L.O., etc.

Conclusion:

This circuitry functions as soon as built, and requires no adjustment whatsoever (apart from adjusting the 2.2MHz trimmer capacitor.) If an L.O. crystal oscillator is used, it should be 'rubbed' (by means of the 50pF trimmer capacitor shown connected between the crystal and ground) to the precise design frequency (in our case 10.490MHz) to keep the frequency shift centered in the passband of the crystal filter.


HIGH-SPEED PHASE-COHERENT FSK GENERATOR - N4ICK -
+5v.
+12v
74LS00
MC12013
MC1496
2·2MHz
TXD IN
RS232 INTERFACE
RTS IN
10,7MH FILTER
10,440 XTAL
TO PIN 10 MC1496
TO +5v
TO +12v
10·7 MHZ FSK OUT

FSK Methods for PACSAT Communication

M. S. Hodgart and Jeffrey W. Ward, GO/K8KA
UoSAT Unit
University of Surrey
UK

ABSTRACT

The authors propose that PACSAT employ several 9600-bit/s noncoherent FSK uplinks and a 9600-bit/s coherent FFSK downlink. This combination of modulation schemes provides for simple groundstation transmitters, groundstation demodulators of several classes of complexity, and staged development of space-rated systems. A research plan is identified which will result in simple spacecraft systems being available quickly and optimised spacecraft systems being developed as time permits. Areas for further study, resulting in 1.5:1 increase in bit rate with no increase in signalling bandwidth, are discussed.

1.0 MISSION PROFILE

Amateur packet radio has expanded greatly in the three years since Den Connors paper "The PACSAT Project" appeared in the 1983 Proceedings [1]. The need for a reliable, high-throughput, world-wide packet service is much greater now than it was 3 years ago; 300-bit/s HF stations cannot continue to provide acceptable long-distance service to exponentially-expanding VHF metropolitan networks. The general objectives and parameters of the PACSAT mission have changed little since the project was first discussed: PACSAT will be a store-and-forward mailbox placed in a polar, low-earth orbit. The mailbox will be served by several 9600-bit/s uplink channels and a single 9600-bit/s downlink. PACSAT will use the AX.25 link-layer protocol, making it compatible with an installed base of more than 10,000 TNCs. Delays in the PACSAT project, caused by lack of funding for the mission, have had some positive results: the volume and power consumption of large RAM devices have decreased, whilst the availability of such devices has increased; current plans call for PACSAT to carry at least 4 Mbytes of message-storage RAM,.

It takes a firm launch opportunity and commitment of funding to solidify the design of any satellite, and in the absence of these stabilizing influences, PACSAT has gone through many design meetings and design revisions. Time has not been wasted, however; experience gained through design, construction and operation of the UoSAT-2 DCE [2] has taught us much that will be of direct use when we begin to work in earnest on a dedicated PACSAT spacecraft. During this time, the UoSAT group at the University of Surrey (UoS) in the UK has become very interested in store-and-forward communications using satellites in low earth orbit. Our interest has moved past the role of supplying a spacecraft "bus" for the PACSAT mission toward actually taking part in the design and construction of the payload. Whatever for-n UoSAT-C takes (and there are several possibilities now being investigated') it will probably carry a PACSAT-like transponder, With this in mind, the authors (with the help of many others both within and outside of UoS) have carried out an investigation of two critical PACSAT design issues: modulation and demodulation, These topics have been discussed by Phil Karn [3], but we feel that enough significant developments have taken place to warrant further investigation,

2.0 LINK BUDGETS

The following discussion is driven by the satellite link budget, as calculated by M. Awan (UoS). This budget assumes that the satellite is to be small and inexpensive, thus dictating a low-power downlink transmitter. The orbital altitude assumed for these calculations is 900 km.

2-Meter Downlink Budge':

Transmitter power	4 W)	36 dBm
Transmit losses		- 2 dB
Antenna gain		0 dBi
EIRP		t34 dBm
Free-space path loss at 145 MHz		-146.3 dB
Carrier power received by groundstation (isotropic antenna)		-112.3 dBm
Noise density for receiver (1 dB noise figure. 300 K equivalent antenna noise temperature.)		-172.9 dBm/Hz

Carrier-power to noise density ratio (l-degree satellite elevation, isotropic antenna.)	60.6 dB/Hz
Bit rate (9600 bit/s)	39.8 dBHz
Available energy per bit divided by the noise-spectral density (Eb/No).	20.8 dB

Rather than assume a perfect system in which this high signal-to-noise ratio is available to a demodulator, Mr. Awan has included an implementation margin in his calculations.

Modem loss	2 dB
Antenna pointing loss	1 dB
Antenna ageing	' dB
Desense due to transmitter	1 dB
Man-made interference	3 dB
Polarisation mismatch	3 dB
Multipath cancellation	1 dB
System Margin	2 dB
	14.0 dB

Subtracting this from the previous result	20.8 dB
	-14.0 dB
Available Eb/No with satellite at 1 degree elevation, with 0dBi-gain receiving antenna.	6.8 dB
Assume that our demodulators need Eb/No to produce a bit error rate (BER) of 1E-6 (as discussed below).	15 dB
Eb/No at demodulator	15 dB
Eb/No available with 0-dBi gain antenna with satellite at the horizon.	- 6.8 dB
Necessary antenna gain.	8.2 dBi

Thus, an 8.2 dBi gain receiving antenna is necessary for 9600-bit/s, 1E-6 BER communication when the satellite is at the horizon. A station so equipped will have excess antenna gain when the satellite is high in the sky, and stations with less gain will have coverage for less than the full horizon-to-horizon satellite pass. Table 1 shows the antenna gain required at elevations from horizon to 90 degrees, as calculated using the above assumptions.

2.1 Possible Downlink on 70-cm?

The above link calculations assume a 2-meter downlink PACSAT could, however, downlink on 70-cm (435 MHz). The difference in free-space path loss between 2-m and 70-cm is about 9.5 dB. Since we must keep the satellite small, this 9.5 dB cannot be made up by simply increasing downlink power or providing antenna gain on the spacecraft. Groundstations would have to increase their antenna gain, but antennas the same size as required for two-meter downlink reception would produce about 3 dB more gain on 70 cm. Additional link "gain" is realized on 70 cm because lower levels of man-made noise reduce implementation loss. While Table 1 shows that stations with 0-dBi antennas would not be able to access the satellite (at the reference BER), stations without antenna-pointing capability could use fixed-pointing gain antennas aimed at high elevations. Thus, using a 70-cm downlink is a possible option for PACSAT and should be explored. To simplify the following discussion, however, we assume that the downlink will be in the 2-meter band.

2.2 Uplink Budget

Whilst free-space loss may make it desirable to keep all PACSAT communications on the lowest frequency available (2 meters), we have discarded this notion for an amateur spacecraft. The state-of-the-art in amateur radio does not easily allow simultaneous transmission and reception within the same band. Receivers do not have enough immunity to front-end overload and transmitters produce wide-band phase noise. The option of having both uplinks and downlinks at VHF might,, nonetheless, be investigated for a store-and-forward satellite operating in some commercial service, where the cost of purpose-built equipment could be justified. In the amateur service we must consider the equipment at hand, and recommend that the uplink be in the 70-cm band, assuming that the downlink is at 2 meters.

Again, we are faced with the 9.5 dB difference in path loss between 2 meters and 70 cm. We assume that, by using methods discussed below, we will be able to make uplink demodulators about 2 dB more efficient than those on the downlink.

Path loss difference between 70 cm and 2 m.	-9.5 dB
Increased efficiency of spacecraft demodulators.	2 dB
Difference between 2-m and 70-cm power budgets.	-7.5 dB

As a point of reference, using uplink antennas having the same gain as downlink antennas (8.2 dBi), the groundstation will need 7.5 dB greater EIRP on 70 cm than the satellite needs on 2 meters, resulting in a requirement for 14 watts transmitter output. This will result in horizon-to-horizon coverage at 9600 bit/s, with BER less than 1E-6. Again, stations with fixed

antennas, lower antenna gain or lower output power would be able to access the satellite at higher elevations or with increased BER.

3.0 MODULATION AND DEMODULATION

Signal-to-noise ratios such as we have been discussing (73 - 15 dB) would produce the desired BER of 1E-6 in systems employing any of several modulation methods. Among the choices are frequency-shift keying (FSK), minimum-shift keying (MSK) and differential phase-shift keying (DPSK). DPSK is described in [3] and has previously been proposed as the modulation method for PACSAT. The major disadvantages of DPSK are that once it has been filtered for bandwidth efficiency it cannot be amplified by limiting amplifiers, and that it requires the use of complex synchronous receivers and transmitters. We propose the use of noncoherent FSK on the uplink and coherent FFSK on the downlink. Noncoherent FSK is simply the technology investigated by S. Goode [4] -- transmitter VCO control voltage is derived from filtered baseband data. Coherent FFSK requires that the phase of the RF signal be strictly controlled, requiring a transmitter more complex than the simple groundstation transmitter. There is, however, an important advantage to using coherent FFSK rather than DPSK: coherent FFSK can be demodulated by simple, non-coherent receiver/demodulators.

3.1 General Uplinks

We propose that groundstations use non-synchronous FSK with a optimum deviation of 3.2 kHz. Modulators would be based on S. Goode's system [4] The data rate will be fixed at 9600 bit/s, and the IF bandwidth of limiting amplifiers fixed at 15 kHz. Receivers on the spacecraft would feed plain quadrature frequency discriminators followed by filters and slicers -- again as in Goode's system. This may seem a technically regressive step, but the very considerable pressures on reliability and power consumption onboard the spacecraft always tend to favor the simplest solution. It is hoped that by developing post-discriminator filters that reduce or eliminate inter--symbol interference, we could realize a 1E-6 BER with uplink signals as low as 13 dB Eb/No. We intend to investigate this in the lab as soon as possible.

The choice of non--coherent FSK for the uplink makes the groundstation modulator/transmitter relatively simple. It also satisfies the desire to have the 9600-bit/s signal fit in the 15-kHz bandwidth of a standard IF. On the 70-cm uplink, maximum Doppler shift will be +/- 10 kHz, and without compensation, this much shift will cause demodulation to fail. We propose that the groundstation be responsible for tracking uplink Doppler to within roughly 500 Hz. In selecting uplink channels, we shall endeavour to provide wide enough spacing that if one groundstation does not correctly track Doppler shift, its transmissions will not drift adjacent uplink channels. Uplink channel spacing of 50-kHz should provide this protection..

There are, of course some disadvantages to using such a simple scheme. First of these is that stations with marginal links cannot decrease their BER by simply decreasing their bit rate. For such a speed reduction to have the desired effect, the IF bandwith of the uplink receivers would have to be narrowed. The complexity of variable-bandwidth receivers is not acceptable. The only way to increase throughput on these links is to resort to forward error correction (FEC) and take advantage of coding gain. P. Sweeney of UoS has investigated simple FEC schemes for store-and-forward satellites. His work indicates that an array code based on two Hamming structures could restore an uplink error rate of 2E-3 to our reference standard 1E-6 BER. His proposed code reduces the data rate to 6600 bit/s (with a signalling rate of 9600 bit/s) and reduces the Eb/No requirement by 3.6 dB. This is a significant benefit to the overall system, allowing the transmitter power to be more than halved.

The other disadvantage to employing simple discriminator demodulators on the satellite is that they are not upward compatible with such desirable types of modulation as Tame FM (also known as Generalized Minimum Shift Keying, GMSK).

3.2 Uplink Enhancements

Given sufficient time, we hope to complete an investigation into a more complex uplink decoder, employing the delay demodulators discussed in [5]. These ingenious demodulators require no clock or carrier recovery, should be highly tolerant of Doppler shift, and could provide virtually the same performance as more complex synchronous decoders (11 dB Eb/No for 1E-6 BER). To take advantage of this enhanced uplink decoder, the groundstation would reduce deviation to +/- 2.4 kHz, but would not have to resort to a coherent transmitter. Unlike a simple quadrature discriminator, a delay decoder could realize decreased BER at lower bit rates (given a fixed groundstation power) without altering receiver IF bandwidth. The uplink signal could still be generated by a Goode-type modem, as it need not be synchronous. Groundstations that do not reduce their deviation will get the same performance that they would have from simple quadrature discriminators. The limitation of the delay decoder is that it is STILL a frequency

discriminator and cannot work on Tame FM/GMSK.

3.3 Research Uplink

Given even more time (and satellite power), we propose to have a "research" uplink for experimentation with synchronous, highly-efficient modulation methods. This uplink would use a De-Buda type synchronous receiver [6] requiring clock and carrier recovery. Stations using such the uplink would HAVE to have phase-controlled coherent transmitters. These increases in complexity are rewarded by the ability to use Tame FM/GMSK and send 14,400 bit/s within the 15-kHz uplink channel. The research uplink would not detract from the PACSAT mission, as the general uplink channels would always provide a standard, predictable service to the user. The research uplink could, however, aid us in the necessary search for higher throughput and provide enhanced service to the advanced user.

3.3 Uplink Summary

We propose that the uplink employ non-synchronous FSK. If spacecraft decoders use straightforward quadrature discriminators, the optimum transmitter deviation is +/- 3.2 kHz. Provided that delay-type demodulators can be developed for the spacecraft environment, groundstations using +/- 2.4 kHz deviation (FFSK) would realize 2 dB advantage over those using other deviations. As a point of reference, a station transmitting 14 watts FSK into an 8.2-dBi antenna would have a BER less than 1E-6 from horizon to horizon. A station with omnidirectional antennas would achieve the same BER while the satellite was above 30 degrees elevation and experience degradation (see Table 1) at lower elevations. Both FEC and transmission-rate reduction should be investigated as ways of providing lower BER to marginal stations. A research uplink, whilst not detracting from the mission, could provide an invaluable tool for development of efficient, high-speed signalling methods.

4.0 DOWNLINKS

It has been demonstrated in the literature [7] (and it has become painfully apparent to packet-radio users on crowded channels) that multiple stations contending for a single communications channel drastically reduce channel efficiency. Thus, as RUDAK [8] serves a 2400 bit/s uplink with a 400-bit/s downlink, PACSAT will be able to serve four or five 9600-bit/s uplinks with a single 9600-bit/s downlink. The modulation method chosen for this downlink must be power and bandwidth efficient and it must yield reasonable results to unsophisticated groundstations. It would also be advantageous if users willing to invest in sophisticated decoders could expect to get better performance for their effort. To fill these requirements, we recommend coherent FFSK, also known as "fast frequency shift keying with spectral modification using non-linear filtering." The deviation at 9600 bit/s will be +/- 2.4 kHz.

The complexity of coherent transmitters is such that we do not wish to insist that users have them. We can, on the other hand, afford the effort to build a few such transmitters for the satellite. coherent FFSK is bandwidth efficient (again we can easily fit 9600 bit/s into 15-kHz) and it produces a constant-envelope signal which can be passed through efficient limiting amplifiers without bandwidth spreading. This is an important consideration when choosing a modulation method for the satellite.

If the satellite transmits coherent FFSK, the user is presented with a range of potential receiver/decoders and an accompanying range of performance. The groundstation need not use a synchronous demodulator - adapted narrow-band FM receivers with discriminator decoders would require 15 dB Eb/No for 1E-6 BER. Delay-type demodulators (theoretically simple to construct) and sophisticated synchronous receiver/decoders should lower this requirement to around 11 dB -- a valuable gain of 4 dB. This is precisely the type of upgradable system that we desire in an amateur-radio operation.

4.1 Research Downlink

We propose that PACSAT carry a research downlink for ongoing experimentation with high-speed, high-efficiency signalling methods. On first consideration, one assumes that if the satellite could power two downlinks, we would be able to double the power on our general downlink. The research downlink, however, would not have the 100 percent duty cycle of the general downlink. It would be turned on only for experiments or for limited "sophisticated user" service. Groundstations using this downlink would have to have synchronous decoders. The data rate could reach 14,400 bit/s within 15 kHz bandwidth or 19,200 bit/s in a 20-kHz bandwidth. To do this we would employ a tighter form of non-linear premodulation filtering -- changing the modulation to "tame FM" (also called Generalized Minimum Shift Keying, GMSK).

4.3 Downlink Summary

Coherent FFSK can be efficiently generated and amplified on the satellite, and provides the users with a wide range of potential decoders. With even a simple discriminator decoder, a user with an 8.2 dB gain

receive antenna could get horizon-to-horizon coverage, while a user with an omnidirectional antenna would get the reference BER of 1E-6 whenever the satellite was above 30 degrees elevation. We would like to experiment with Tame FM (GMSK), which would allow us to increase bit rate by half without increasing uplink bandwidth. Such research should be accornodated in the final PACSAT design.

5.0 DOPPLER TRACKING

In the assumed 900 km orbit, maximum Doppler shift at 2 meters is +/- 3.5 kHz, and at 70 cm it is +/- 10 kHz. While some frequency error between transmitter and receiver has been accounted for in the link calculations under implementation margin, we believe that error much greater than 500 Hz should be avoided. We need to do further research to support this claim. The coherent FFSK proposed for the downlinks can, with suitable data randomization, yield a DC-free baseband signal. Any dc level on the demodulated signal would then indicate frequency error. This dc "error signal" becomes the basis for Doppler tracking.

It was originally proposed that the uplink receivers track Doppler shift, allowing the user to set his transmitter anywhere in a wide uplink channel and not change frequency over the course of a pass. We recommend that this technique be discarded. Our recommendation is based on the following scenario: Imagine that two users, one "in front of" the satellite and the other "behind" the satellite are both sending packets on the same uplink channel. One of these users is seeing nearly +10 kHz Doppler shift, while the other sees nearly -10 kHz Doppler (in the worst case). If the uplink receiver is responsible for Doppler tracking, on alternating packets it will have to swing 20 kHz to lock on the necessary signal. As far as we can tell from the literature and from personal contacts, no one has been able to track this much instantaneous frequency shift at a data rate near 9600 bit/s. If we were to use AFC loops with such bandwidth, they would probably take hundreds of bit periods to lock up, cutting into precious uplink communication time.

We propose that the groundstation be responsible for tracking Doppler shifts. If the station is computer-controlled, the computer can easily produce Doppler information in analog or digital form for the uplink transmitter. If the station is not computer controlled, the downlink receiver must already have some closed-loop Doppler-tracking mechanism, which could feed a tracking signal to the uplink transmitter. For the groundstation receiver, tracking the spacecraft is a relatively easy task. The groundstation sees a smooth Doppler shift over the course of a pass, not the rapid switching that would have to be tracked by the satellite. Development of suitable groundstation receivers and transmitters should, we believe, be undertaken or at least coordinated as part of the PACSAT project.

6.0 ADAPTIVE COMMUNICATION

Notwithstanding that the realization of a single-speed, 9600-bit/s, space-engineered communications system will be a major task, we propose that PACSAT have an optional reduced data rate to accomodate poor links and/or an optional high data rate for especially good links. The importance of these options can be understood by studying Table 1. For our proposed PACSAT orbit, the free-space path loss is 12 dB less when the satellite is overhead than when it is at the horizon. A station with steerable antennas, equipped to communicate with the satellite at low elevation angles will have 12 dB "extra" link margin when the satellite is overhead. This extra margin could easily accomodate an increased data rate while the satellite is high in the sky. The free-space loss profile of the satellite-groundstation link allows stations with omnidirectional 0-dBi antennas to the satellite when it is 30 degrees or more above the horizon, but this is not the most efficient solution for stations with fixed antennas. These stations need antennas adapted to the link profile --- 8 dBi on the horizon and down to -3 dBi overhead. Alternatively, a station equipped with high-speed encoders and decoders could USC a directional antenna fixed at a high elevation, communicating all of its traffic in a bandwidth-efficient manner when the satellite was closest. Such adaptive communications are well suited to the packet-radio environment, in which administrative messages can be communicated between satellite and groundstation without disrupting other communications. Users may be able to "bargain" with PACSAT for an increased or decreased communications rate.

7.0 CONCLUSION

Presented here are some thoughts concerning PACSAT design. Our recommendations are NOT official PACSAT design decisions. They are presented to give users some idea of how we arrive at design decisions, to give already idle technicians some things to implement, and to allow those with opinions on these matters to express those opinions. We cannot hope to decide today what will be the best modulation methods available in the future, but we think that FSK and FFSK will provide PACSAT with the communications tools needed to carry out its mission. The proposed combination of coherent and noncoherent methods should provide a smooth

transition between the relatively slow data communications available to amateurs today and the fast communications that will have to become available in the near future.

8.0 ACKNOWLEDGMENTS

This paper is not simply the work of the two authors, but the result of many discussions, including those with Richard MacBeth (G8VLY, UoS), Martin Sweeting (G3YJO, UoS), Peter Sweeney (UoS), Harold Price (NK6K), and R. Gibson (Phillips Research Lab).

TABLE 1 - PATH LOSS and REQUIRED GAIN VS. SATELLITE ELEVATION

El.	Path Loss	Gain on 2 meters	Gain on 70 cm.
0	146.6	8.2	17.9
10	143.9	5.5	15.0
20	141.5	3.1	12.6
30	139.5	1.1	10.6
40	137.9	-0.5	9.0
50	136.7	-1.7	7.8
60	135.8	-2.6	6.9
70	135.2	-3.2	6.3
80	134.9	-3.5	6
90	134.8	-3.5	6

(1) Elevation of satellite in degrees.
(2) Free space loss at 2 meters.
(3) Gain (dBi) necessary to achieve 1E-6 BER on 2-meter downlink.
(4) Gain (dBi) necessary to achieve 1E-6 BER on 70-cm downlink.

REFERENCES

[1] "The PACSAT Project," Den Connors, Proceedings of the Second ARRL Amateur Radio Computer Networking Conference, ARRL, 1983.
[2] "The UO-11 DCE Message Store-and-Forward System," H. Price, J. Ward, Proceedings of the Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986.
[3] "Modulation and Access Techniques for PACSAT," P. Karn, Proceedings of the Second ARRL Amateur Radio Computer Networking Conference, ARRL, 1983.
[4] "Modifying the Hamtronics FM-5 for 9600 Bps Packet Operation," S. Goode, Proceedings of the Fourth ARRL Amateur Radio Computer Networking Conference, ARRL, 1985.
[5] "Differential detection of MSK with non-redundant error correction," T. Masamura, et al, IEEE Transactions on Communication Vol 25, No 6, June 1979.
[6] "Coherent demodulation of frequency shift keying with low deviation ratio" R. De Buda, IEEE Transactions on Communication, June 1972.
[7] Computer Networks, A. Tanenbaum, Prentice-Hall, 1981.
[8] "Formal Definition Meeting for the Packet Radio Experiment RUDAK to be Included in AMSAT P3-C," Proceedings of the Fourth ARRL Amateur Radio Computer Networking Conference, ARRL, 1985.

A Packet Controller for the Revolution

by Lyle Johnson, WA7GXD
co/ Tucson Amateur Packet Radio
PO Box 22888
Tucson Arizona 85734-2888

Tucson Amateur Packet Radio (TAPR), an Amateur-based, volunteer organization, is dedicated to the advancement of Amateur digital communications. Its first widespread TNC design, now called TNC 1, is in use throughout the world.

In spite of its wide acceptance, the high cost of the TNC 1 (about $300 as a kit and $500 assembled> discouraged many Amateurs from entering the exciting world of packet radio.

The TNC 2 project was launched with two primary objectives: reducing the cost of entry into Amateur packet radio while maintaining the high standards set by the TNC 1.

Any list of high-cost items used in the TNC 1 must include the WD1933/1935 HDLC controller, the XD2212 NOVRAM (tm) and the cabinet. Together, these items represent nearly $100 of the $320 cost of the TAPR TNC 1 kit.

In addition to eliminating the above components, careful attention was directed to parts or features that could be deleted from the new TNC design without seriously affecting performance. As a result of this, TNC 2 lacks a software-programmable baud rate generator, user parallel ports and a programmable memory map ROM.

Finally, ways were explored to enhance the operation of TNC 2. Hundreds of letters have been received by TAPR since the beta testing of the original TNC design, and many of the suggestions contained in those letters were implemented in the new unit.

TNC 2 is based on the Z80 (tm) microprocessor (uP). In addition to its efficient interrupt structure, the wide availability of software tools for this device make the TNC 2 more suitable for the experimentally inclined Amateur than TNC 1, which is based on the less common 6809 processor.

TNC 2 uses an SIO dual channel serial controller: one channel serves as the user asynchronous port, while the other serves as the HDLC controller. Both serial ports are full-duplex. The user port output is buffered to RS-232 compatible levels by a low-power op amp. Incoming RS-232 level data is translated to TTL logic levels by a CMOS Schmitt trigger IC.

In order to use the SIC1 on Amateur packet channels, a means had to be devised to convert the NRZ (Non Return to Zero> format of the SIO's HDLC data to and from title NRZI (Non Return to Zero-Inverted> format. This problem was solved by use of a single flip-flop on transmit and a two-chip state machine for the receive side,

As with other logic on the TNC 2, the NRZ-NRZI circuitry was implemented with ICs readily available in CMOS technology. CMOS ICs typically consume far less power and have much greater tolerance to noise than other types, and are now speed and cost competitive with earlier, less optimal ICs.

The TNC 1 has six byte-wide memory sites mapped for 8k byte parts. This results in 48k of available memory without reprogramming the address decoder. The release software (pre-version 4.0) utilized only 40k of this memory space. TNC 2, on the other hand, has only three bytewide sockets. In the final configuration, one socket is mapped for a 32k byte EPROM and the other two sockets can use either a pair of 8k byte RAMs or a single 32k byte RAM. Thus, TNC 2 normally supports 48k bytes of memory and can easily accomodate 64k bytes, the entire address capability of the processor used.

To solve the problem of memory volatility (the loss of data upon the interruption of power), a battery-backed RAM (bbRAM) scheme is employed for all RAM on TNC 2. This not only eliminates the need for NOVRAM, as used on TNC 1, it provides greatly expanded nonvolatile storage.

To simplify the visual interface, the number of LEDs on the TNC front panel was reduced from eight to five. At the same time, more useful information is presented. LEDs are provided for POWER, PTT (indicating transmitter activation>, DCD (data carrier detect - indicating that the modem has detected incoming data), STA (status - indicating unacknowledged frames in the TNC's buffers) and CON (connected -- indicated the TNC is in the connected or disconnecting states).

An effort was made to enable TNC 2 to operate from a single source of DC power Cnominally 12 volts with negative common1 for portable and emergency use. However, a decision was made early on to assure

that the RS-232 serial port output was capable of driving a standard RS-232 load to below -3 volts. This meant a negative voltage source was required.

This was provided by means of a full-wave charge pump circuit based on a dual 555 timer IC. Unregulated +12 and -V are used for the RS-232 driver. Regulated +5 and -5 voltas are derived from the regulated sources. The modem ICs, which require more than 9 volts of DC and are somewhat voltage sensitive, are driven from the +5 and -5 volt sources. Simple bipolar transistor level shifters are then employed to interface the modem to the TTL levels required by the SIO chip and NRZ-NRZI conversion circuitry. Not coincidentally, the MF-10 switched-capacitor filter IC used in the receive portion of the modem requires +5 and -5 volt sources.

As with the earlier TNC, TNC 2 includes on-board circuitry to aid in modem calibration. The modem parameters are configurable via plug-in DIP headers. Switch selectable data rates for the radio channel include 300 and 1200 baud, which are the current HF and VHF packet standards. A higher-speed clock for 9600 baud operation is also included, but requires the use of an off-board modem. The modem disconnect is virtually identical to that of TNC 1, and the K9NG 9600 bps modem has been successfully interfaced to both TNC 1 and TNC 2.

The TNC 2 was introduced at the 1985 Dayton Hamvention at a cost of well under $200. Available to the general Amateur community in August of the same year, the intial production run of 1200 units was quickly exhausted.

At that time, TAPR determined to step out of the TNC marketplace. Production rights were licensed to several commercial firms, allowing TAPR to concentrate its efforts on development of otherwise unavailable packet devices, including a networking controller and high speed RF deck.

Licensed versions of TNC 2 are now available from various sources at prices between $100 and $200. Truly, participating the packet revolution is now within the financial grasp of virtually every radio Amateur.

Acknowledgmenta

While a complete list of all the volunteers who helped with suggestions or testing of the TNC 2 would be quite lengthy, it is appropriate to include the names of the prime movers of the project, without whose help TNC 2 would not have occurred.

Peter Eaton	WB9FLW	PC Board Layout
Howard Goldstein	N2WX	Software Design
Steven Goode	K9NG	Modem
Charles Green	NOADI	Documentation
Daniel Morrison	KV7B	Testing
Margaret Morrison	KV7D	Documentation
Paul Newland	AD71	Hardware Design
Harold Price	NK6K	Protocol

TNC 2 PARAMETER SETTINGS AND MEANINGS

Thomas A. Moulton, W2VY

The Radio Amateur Telecommunications Society
206 North Vivyen Street
Bergenfield, NJ 07621
(201) 6874630

ABSTRACT

This paper will describe the command set for the TNC 2 and help the reader set the parameter values.

INTRODUCTION

The commands will be discussed in groups of related commands. The groups are: Terminal, Monitor, Link, Command, Special features and Utilities. The values that parameters should be set to will fall into four catagories, Don't touch, Set and forget, Experimental and As you wish.

TERMINAL INTERFACE

The first parameters that will be set from this group are: PARITY, AWLEN, ECHO, and AUTOLF.

The default parity is even with 7 bit bytes. First try to match these with your terminal or terminal program. If this still gets errors, garbage characters or error messages from your terminal program, try for no parity, (PARITY 0 or 2) and MARK or SPACE or NO parity in the terminal program.

The ECHO is easy, if you don't see what you type turn echo ON (Default). If you SSEEEE DDOOUUBBLLEE turn it off. The AUTOLF is used to add a line feed after the carriage return sent to the terminal so the lines don't over print. If the text is double spaced then AUTOLF needs to be turned off.

If you are using a printing terminal you may have to set a few special parameters dealing with how physical printing terminals work. The older printers need time to physically move to perform a linefeed or carriage return. To create this delay it is common pratice is to send the ASCII NUL character ($00). To add nulls after a linefeed turn NULF on; to add nulls after a carriage return (most common) turn NUCR on and set the number of nulls with the NULLS command.

To insure that long lines of text get printed correctly, you should set SCREENLN to the width of your terminal or printer. Many CRTs have special functions that are selected with sequences that start with ESC ($1B). To avoid this you should set ESCAPE ON. Some old terminals do not support lower case. This is not a real problem. If you turn LCOK OFF then all text sent to the terminal will be converted to upper case. With most printers you will want to turn BKONDEL OFF so you can read what you are typing even if you backspace over it. This will print a \ for each backspace you type instead of backing up and replacing the unwanted character with a space,, This is up to your own personal preference.

There is a command to reprint your terminal input buffer. This command is set by the REDISPLAY parameter. The default is $12 which is a CTRL-R (CONTROL KEY and R KEY depressed at the same time, the control key is like the shift key). To delete the last character typed you can type a CTRL-H (DELETE OFF) or a RUBOUT (DELETE ON). Usually one of these is what is sent by a terminal when the BACKSPACE, RUB, DELETE or left arrow key is typed. It is usually found in the upper right portion of the keyboard (Not part of the numeric keypad). If you want to delete the entire input buffer you can type the CANLINE command, which is usually set to CTRL-X.

SENDPAC is the command that forces the data in the input buffer to be placed in a data frame and on the out-going queue. This is normally set to CR ($0D). A frame will also be formatted and queued when the input buffer gets more than PACLEN characters in it. This can be used to your advantage, if you set SENDPAC to CTRL-A you can then type many lines in one frame. This will reduce the number of packets you send. When you do send, you will generally send full packets. If you then type CTRL-X you will only delete the last line you typed. If you type the CANPAC character you will cancel the current packet, (i.e., just the packet that hasn't gone to the output queue yet). If you use this mode you may want to turn CR off, this will make the TNC not add the sendpac character to the out going data. In

some cases you may need to add a linefeed after CR in the outgoing data, you can do this by turning LFADD on.

When the TNC is sending to you too fast, you can stop it by typing the STOP character (CTRL-S) and you can restart it with the START character (CTRL-Q). If you need to type any of these characters you can use the PASS character (default CTRL-V), sometimes refered to as the quote character. For instance, if you wanted to send a CTRL-A you, would have to type CTRL-V CTRL-A; obviously, to send a CTRL-V you would type CTRL-V CTRL-V.

MONITOR

The easiest way to find out wh0 in your area is on packet or what BBS systems are around is to monitor the channels, mostly 145.010 MHZ but also 030, 050, 070, 090, as we11 as 220 MHZ in some areas.

The default monitor mode serves well for this purpose. You may want to change some of the parameters once you know what facilities are in your area. To only see beacon messages turn MALL OFF. If you want to watch all types of frames you can turn MCOM ON. If you want to monitor while you are connected you can turn MCON ON (if you really want!).

There are two commands to help the formatting of the monitor data. To remove the digipeat path turn MRPT OFF, you can also have the frame header and the text on seperate lines if you turn HEADERLN ON. If the real-time clock is set you can have the date and time as part of the header with MSTAMP ON.

COMMANDS

The first thing you MUST change is MYCALL, set it to your callsign. Now you can attempt to establish a link to another station. Select the strongest digipeater you hear and try to connect to yourself. If the digipeater in your area is W1AW-2 then you should type C mycall VIA W1AW-2; if everthing is working correctly you should be connected to yourself. When you get the ***CONNECTED message you will change from command mode (with the cmd: prompt) to the mode determined by the CONMODE parameter, the default is CONVERSe mode. You can type and talk yourself (exciting huh?). When you are done, disconnect by typing the COMmand character (CTRL-C). You will receive the command prompt (cmd:) and should enter D (DISCONNECT). After a few seconds you will get the message ***DISCONNECTED. You can connect to other stations in a similar fashion, see the manual for more details.

If you want to change the command character, to the ASCII ESC character, enter COMmand $1B. If you want to go from cmd mode to convers mode enter CONVers. The NEWMODE command will have the TNC re-enter command mode when the link disconnects if it is set ON. If NOMODE is ON, the only way to change modes is by explicit command.

When you are using a BBS, you may find that you are able to connect directly to it, but when activity increases you need to use the digipeater. Instead of having to disconnect and reconnect, you can enter command mode and issue a RECONNECT command with a more reliable path.

While you are in command mode received data will be printed only when you are not typing, this can be overridden by setting FLOW OFF. If you want to verify what you have typed you can hit the Redisplay command (CTRL-R) to see your input buffer.

When sending data to the TNC, such as sending a file, the TNC should be able to flow control (or stop) the computer. There are two ways of doing this. If XFLOW is on it will use XON and XOFF as start and stop characters for the data going to the TNC, as START and STOP were for data going to the terminal. If XFLOW is off, the TNC will use hardware flow control, using the RS-232 RTS (Request To Send) and CTS (Clear To Send) to start and stop data.

For file transfers that are not strictly text files the transparent data mode is used, in this mode there is no line editing or delete. You can enter transparent mode with the TRANS command, or by having Conmode set to Trans. There are two ways to return to command mode while still connected, send a break signal to the TNC, or by entering three COMMAND characters in a row with idle time before and after them, a delay of more than CMDTIME seconds. This is like the common smart telephone modem attention signal.

To have the most transparency you should run with TRFLOW and TXFLOW both OFF, which means you are using hardware flow control both ways. When using the software flow control you must use the PASS character (CTRL-V) to send the XON or XOFF as data, which is a problem for most terminal programs.

In many cases 100% transparency is not really needed, some word processing files only need to insure 8 bit data and don't use many control characters. To provide for this and allowing the local editing you can set AWLEN 8 and 8BITCONV ON.

LINK PARAMETERS

The timers associated with the links can make or break the network.

The TXDELAY is the time required to get your transmitter to send a stable signal plus the time for most receivers to produce audio plus the time required for the modem to detect your tones. The default is usually ok, but please feel free to adjust it if needed, the shorter the better.

If you are using a voice repeater you will then need to set AXDELAY to account for the key up time of the repeater. The next timer comes after the data is sent, FRACK is a gauge of when the acknowledgement should return. This is based on the number of intermediate stations. FRACK should be left alone at this time. There may be some networks where it should be changed, leave that to the local management bodies to investigate.

The timers for deciding when to send are also very important. When listening to the channel there is a minimum time your TNC must wait before it can transmit. This is called DWAIT, or Digipeater WAIT time, it is to give the digipeater a chance to repeat a packet. The value this is set to depends on what your TNC is going to be used for. When digipeating no wait is included. For interactive QSOs the default of 16 (160 ms) is good. The lower priority traffic, such as file transfers and BBS sites should have a longer DWAIT. It is suggested that 240 ms and 320 ms be used for these types respectively.

When the DWAIT has been satisfied, your TNC then can start a transmit sequence. The transmitter is assumed to be on after TXDELAY has elasped. If DWAIT + TXDELAY is greater than AXHANG then the TNC must transmit to satisfy the AXDELAY. Then the data can be sent.

When operating in full duplex, FULLDUP ON, the DWAIT is ignored, since you are transmitting on a different frequency.

The maximum ammount of data contained in a frame is set by PACLEN. The default is 128, but it is perfered that 256 byte data fields be used (PACLEN 0). The larger packet sizes can be taken advantage of if you only forward data when you have a full packet or new data is not being typed. To do this you can set SENDPAC $01 along with CR OFF and CPACTIME ON. PACTIME specifies when data should be forwarded. There are two options: PACTIME EVERY n, forward data every n seconds and PACTIME AFTER n, forward data after n seconds of no new data. The latter is the more commonly used form, for BBS and other computer generated data.

The PASSALL option is generally not usefull, except for debugging. Ignoring the CRC defeats mast of the purpose of having an error free link.

The RESPTIME parameter is a timer used to help insure that a station does not transmit an acknowledgement too early. It's default is 1.2 seconds which should be fine for all cases.

The RETRY counter is important. It should be short for BBS forwarding (3 at the most), the concept is that if the link is shakey don't bother causing more interference. It should be longer for users and user initiated file transfers. The TRIES command can be used to view and optionally change the current retry counter, without changing the value of RETRY.

The CHECK timer is used to keep in touch with the true status of your link. If no packets are received from the station at the other end of the link within the time specified by CHECK parameter, then the TNC will take actions based on the AX25L2V2 flag. If AX25L2V2 is ON then the TNC will send a supervisory packet (RR) to verify that the other station is still connected. If no acknowledgement is received within the retry counter number of tries or if the AX25L2V2 flag is off your TNC will send a DISC frame to clear the connection.

The MAXFRAME parameter sets the maximum number of data frames that can be outstanding at a given time. The default is 4 and should be fine for most applications.

The UNPROTO is a destination and digipeater path that beacons and other data sent while not connected are routed to. To enable your transmitter you should set XMITOK ON.

The MYALIAS command is an interesting one. It can be used to set a call or name to be used as a digipeater. The alias can only be used for digipeating. We could use RATS as an alias, in order to keep legal you must insure that you properly identify the station, this can be done with HID ON, which will send an ID frame every 9.5 minutes of activity. An ID frame is simply a frame with the TNC call/R sent to the UNPROTO address.

UTILITY COMMANDS

The real-time clock is based upon the CPU clock. Since they vary, the

CLKADJ parameter is used to compensate for the difference. You should adjust the clock speed (aka crystal frequency) to keep most of the spurs away from the packet channels as much as possiable. Many times Murphy will put the spur right on 145.010 MHZ!

The clock is set with the DAYTIME command in the format YYMMDDHHMM. The date can be displayed in two formats depending on the setting of DAYUSA. If DAYUSA is on the date is in the form mm/dd/yy. If it is OFF the format is dd-mm-yy.

The TRACE mode is used for debuging problems between TNCs. It is useful1 if you are familuar with the protocol and various formats. The frames are broken down in Hex and ASCII.

The internal modem is aligned with the aid of CALSET and CALIBRATE commands. Full details on calibration is in the operators manual for the TNC.

The DISPLAY command is used to display sets of or all the parameters. The RESET command will reset the bbRAM to the defaults, all customization will be lost. The RESTART command will perform a power on reset.

SPECIAL FEATURES

Some of these features are very basic and don't really fit under this catagory, but they are higher level functions and belong as user services or features that you could do by hand or with a computer.

The MHEARD list is a list of stations that frames have been heard. The date and time is recorded. This list can be cleared with the MHCLEAR command and is displayed by entering MHeard.

The BEACON option should not be used on a regular basis. It causes unwanted interference and are not needed.

When a station connects to your TNC you can have a canned message sent to them if CMSG is ON. A common message is CTEXT I'm not here now leave a message in my buffer. YOU can have the connect messages time stamped if the clock is set and CONSTAMP is ON.

When your station is operating in unattended mode, you might want to ignore frames from the local BBS. You can do this by using BUDLIST OFF and LCALLS set to the BBS callsign. These options can be very useful1 in filtering unwanted data from your capture buffer.

The default setting of the TNC only allows one connection at a time. This can be changed by changing the number of USERS. The default is 1. When it is any other value an incoming connection request is assigned the lowest free stream or port. Ports are numbered from A to J. The STREAMSW command sets the character that is used to mark a stream identifier. The default is $7C (!). When new data is received it is marked with the stream switch character followed by the stream identifier (A-J). This may also include the stations callsign if STREAMCA is ON. In order to help keep it clear where a stream begins you can have the TNC double the stream switch characters it prints, (ex. !!A:W2VY: hi there). The stream switch character is also used to specify which you want to send a line to, (ex. !A Hi Tom).

Any of the links can be= made perminate with the CONPERM option. This is stream dependent, if this is set ON the link will never enter the disconnected state. The CSTATUS command is used to display the status of all streams. Including a P if the link is marked as perm.

SUMMARY

The commands are summarized on the following page, including the default values. Some of these parameters may be removed as more powerful1 networking protocols are developed for the TNC 2. New commands and features will be added.

The TNC 2 and it's clones are going to be the "Smart modems" of radio. With the larger population of packeteers packet radio is going to be advancing very quickly in the next year. Enjoy!

Command -w-m---	Default -------	Description
8bitconv	OFF	Strip high-order bit when in convers mode
AUtolf	ON	Send Linefeed to terminal after each CR
AWlen	7	Terminal character length (7/8)
Ax25l2v2	OFF	Run as version 1.0 of AX.25
AXDelay	0	(O-180 * 0.1 sec) Voice Repeater keyup delay
AXHang	0	(O-20 * 0.1 sec) Voice Repeater hang time
Beacon	E O	Every/After O-250 *10 sec.
BKondel	ON	Send BS SP BS for each DELETE character

BText		(120 char) Text to be sent for a beacon
BUdlist	OFF	Stations in Lcalls are ignored
CALibra		Used to calibrate the builtin modem
CALSet		Used with CALibrate
CANline	$18	(Control-X) The Line Delete character
CANPac	$19	(Ctrl-Y) Cancel current character
CHech	30	(0-250 * 10 sec) Idle link time out
CLKADJ	0	(0-65535) Real time clock adjustment constant
CMDtime	1	(0-255 sec) transparent mode escape timer
CMSG	OFF	Don't send CTEXT when user links to your **TNC**
COMmand	$03	Char to escape from CONVERS mode to command mode
CONMode	CONVERS	(or TRANS) Mode to enter when link established
Connect		Establish Link with station via optional stations
CONOk	ON	Allow stations to establish a link with your **TNC**
CONPerm	OFF	If ON always keep this link up (never Disconnect)
CONStamp	OFF	If ON print date & time stamp on connect messages
CStatus		Prints the status of all links (Streams)
CONVers		Enter Converse mode from command mode
CPactime	OFF	Don't forward data based on timers (see Pactime)
CR	ON	Append a Carriage Return to each data packet
CText		(120 Ch) Connect Message Text (see CMSG)
DAytime		Date and time for real time clock
DAYUsa	ON	Print date as mm/dd/yy instead of dd-mm-yy
DELete	OFF	The character delete is BS ($08) not DEL ($7F)
DIGipeat	ON	Allow stations to use you as a Digipeater
Disconne		Request a link disconnect from the other station
Display		(Async/Character/Id/Monitor/Timing) Parameters
DWait	16	(0-250 * 10 msec) Delay to let digipeater repeat
Echo	ON	Echo characters typed on keyboard to terminal
EScape	OFF	Don't translate ESC character ($1B) to $ ($24:)
Flow	ON	Don't print to terminal while user is typing
FRack	3	(1-15 sec) Time needed to ack a packet per station
FUlldup	OFF	Operate in Simplex mode
HEaderln	OFF	Print the frame header and text on the same line
HID	OFF	Don't send an ID packet every 9.5 mins when active
ID		Force an ID packet (UI frame Via UNproto path)
LCALLS		(0-8 calls) to receive or ignore stations (BUDLIST)
LCok	ON	Do not convert lower case to UPPER CASE on terminal
LCSTREAM	ON	Convert the stream select specifer to Upper case
LFadd	OFF	Add a Line Feed after each CR send to the terminal
MAll	ON	Monitor data frames as well as beacons
MAXframe	4	The window size for outstanding frames
MCOM	OFF	Monitor only data frames instead of all types
MCon	OFF	Don't monitor frames when linked to another station
MFilter		Up to 4 characters to be removed from monitored data
MHClear		Clear the calls Heard list
MHeard		Display the calls heard and date/time if clock set
Monitor	ON	Monitor mode on - see BUDLIST, MALL, MCON, MSTAMP
MRpt	ON	Display the digipeater path in monitored frames
MStamp	OFF	Monitored frames are Not time stamped
MYAlias		An identifier for a digipeater
MYcall	NOCALL	The station callsign for ID and linking
NEwmode	OFF	The TNC acts like a TNC 1 for changing modes
NOmode	OFF	If ON allow explicit mode change only
NUcr	OFF	Don't send NULLS ($00) after a CR
NULf	OFF	Don't send Nulls after a LF
NULLS	0	(0-30) Number of nulls to send as requested
Paclen	128	(0-255,0=256) size of the data field in a data frame
PACTime	After 10	(Every/After 0-250 *100 ms) Data forwarding timer
PARity	3 (even)	(0-3) Terminal parity 0,2=None 1=odd 3=even
PASs	$16	(CTRL-V) char to allow any character to be typed
PASSAll	OFF	Accept only frames with valid CRCs
RECOnnect		Like Connect but to restablish a link via a new path
REDisplay	$12	(CTRL-R) char to print the current input buffer
RESET		RESET bbRAM PARAMETERS TO DEFAULTS
RESptime	12	(0-250 * 100 ms) minimum delay for sending an ACK
RESTART		Perform a power on reset
RETry	10	(0-15) maximum number of retries for a frame
Screenln	80	(0-255) Terminal output width
SEndpac	$0D	(CR) Char to force a frame to be sent
STArt	$11	(CTRL-Q) the XON for data TO the terminal
STOp	$13	(CTRL-S) the XOFF for data TO the terminal
STREAMCa	OFF	Don't show the callsign after stream id

STREAMDbl	OFF	Don't print the stream switch char twice (!!A)
STReamsw	$7C (!)	Character to use to change streams (links)
TRAce	OFF	Hexidecimal trace mode is disabled
TRANS		The TNC enters Transparent data mode
TRFlow	OFF	Disable flow control to the Terminal (Trans mode)
TRIes		(0-15) set or display the current retry counter
TXdelay	30	(0-120 * 10ms) Keyup delay for the transmitter
TXFlow	OFF	Disable flow control to the TNC (Transparent mode)
Unproto	CQ	Path and address to send beacon data
USers	1	Sets the number of streams (links) allowed
Xflow	ON	XON/XOFF Flow control enabled instead of hardware
XMitok	ON	Allow transmitter to come on
XOff	$13	(CTRL-S) Character to stop data from terminal
XON	$11	(CTRL-Q) Character to start data from terminal

Link Level Protocols Revisited

Phil Kam, KA9Q

Brian Lloyd, WB6R QN

ABSTRACT

The LAPB protocol on which the connected mode of AX.25 is based was originally designed for point-to-point wire links, not shared multiple-access radio channels. This paper discusses the deficiencies of LAPB in the radio environment and suggests several improvements. These) include the simple (adjusting existing TNC parameters), the moderate (upward compatible implementation changes) and the radical (replacing LAPB altogether with a simpler and inherently much more efficient protocol).

We believe that these approaches deserve serious attention by the amateur packet community. The suggested AX.25 congestion control techniques should be used as soon as possible in existing networks, while the new "ACK-ACK" protocol should be considered in the design of backbone networks and eventually user-network links.

1. AX.25 in Review

The Amateur Radio Link Layer Protocol AX.25 [1] actually consists of two distinct sublayers. The upper sublayer is a connection-oriented protocol almost identical to the Link Access Procedures Balanced (LAPB) from CCITT X.25 [2]. Prepended to the LAPB control field in AX.25, however, is a special address field containing ASCII-encoded amateur radio callsigns and substation IDs (SSIDs). Each AX.25 frame contains, at a minimum, the callsign/SSID of the sender and intended receiver of the frame. Optionally, it may also contain a sender-specified list of digital repeaters, or digipeaters, through which the frame should be routed to its destination.

This additional lower sublayer contains most of the changes made to enable it to function in an amateur radio environment. An AX.25 address header always contains full source and destination addresses. Therefore it meets the definition of a "datagram" protocol, and we will call this field the "datagram sublayer" of AX.25.

The LAPB (upper) sub-layer of AX.25 belongs to the general class of ''ARQ'' (automatic request-repeat) protocols. It uses receiver acknowledgements (''ACKs'') along with sender timeouts and retransmission to increase the reliability of the service provided to higher layer protocols above that of the raw datagram sublayer.

The performance of an ARQ protocol depends heavily on how closely its design assumptions are met in practice. LAPB was designed for a full-duplex, point-to-point channel with low bit error rates; thus it performs poorly when used on a half-duplex, multiple-access radio channel with a high error rate. The remainder of this paper will discuss the special requirements of the amateur environment, analyze LAPB with respect to these requirements, and propose several alternatives. These range from a set of backward-compatible provisions to improve the LAPB retransmission algorithms in the presence of channel congestion, to the specification of an entirely new, connectionless link level control protocol that is both easier to implement (especially in the "multiple connect" case) and inherently much more efficient than LAPB.

2. Packet Radio Protocol Requirements

A good packet radio protocol must do four things:

1. Deliver the user's data as reliably as possible.
2. Deliver the user's data as quickly as possible.
3. Use the shared channel capacity as efficiently as possible.
4. Be reasonably easy to implement.

Naturally, these goals conflict. For example, on a full-duplex, point-to-point channel, the third consideration does not apply. The protocol is free to use as much transmission bandwidth as it chooses to maximize the other criteria, since it would otherwise go to waste. On a shared packet radio channel, however, a socially well-adjusted protocol should attempt to use as little channel bandwidth as possible to move a given amount of data, even at the expense of increased user delays. As we will see, this makes several features of LAPB undesirable.

3. Sliding Windows vs Stop-and-Wait

LAPB is a *sliding window* protocol. This means that it is possible to send more than one frame before an ACK is received for the first outstanding packet. The maximum number of frames that may be transmitted before the sender must stop and wait for an ACK is known as the window size. For the standard form of LAPB used in AX.25, the window size cannot exceed 7. Most implementations allow the user to decrease this limit further (e.g., the TAPR MAXFRAME option).

3.1. Pipelining

Sliding window protocols are popular because they allow the full utilization of high speed, *full duplex* circuits with long round trip delay. For example, a satellite path has a round trip delay of .5 seconds. If only one frame can be sent at a time, the transmitter must remain idle at least .5 seconds before an ACK could return, enabling it to transmit another frame. While the effect of this delay can be reduced by making each frame very long it cannot be eliminated unless more than one frame can be "in the pipe" at a time.

Pipelining causes major problems, however when the channel is half duplex. Since the receiver usually attempts to acknowledge incoming data as soon as the channel becomes clear, it is likely to collide with the sender if the sender attempts to send additional data without first waiting for the ACK for the data already sent.[1] Pipelining on a half duplex radio channel is therefore a. good example of a socially undesirable technique that attempts to reduce user delay at the expense of overall channel efficiency.

3.2. Transmission Efficiency

Given that a user wants to send a certain amount of data, a packet radio controller must control several parameters to packetize that data for transmission most efficiently.

Since it operates on a half-duplex channel, the controller must first decide how much to send on each transmission before switching to receive and waiting for an ACK. Sending more on each transmission reduces channel overhead by allowing each returning ACK to "cover" more data. It also increases the maximum throughput attainable on a channel with a given propagation dellay, since this rate can never exceed one window full per round trip interval. On the other 'hand, smaller transmissions are smaller targets for noise and interference.

Second, once it has decided to send a certain amount of data in a transmission, the data must be divided up into one or more consecutive frames.[2] The fewer frames used, the smaller the effects of frame overhead. On the other hand, a single large frame is unusable even if a single bit error occurs near the end, so sending the data as a burst of smaller frames might allow the receiver to "salvage" at least the beginning of the transmission. (Note that the "go-back-N" recovery strategy of LAPB requires that all frames received after an error be discarded and retransmitted, even if the later frames are received correctly).

The sender therefore controls three parameters: transmission size, frame window

[1] Note that TNCs capable of "multi-connect" operation (i.e., able to manage simultaneous connections with several different stations on the same channel) should allow unacknowledged data on only one connection at a time.

[2] Although pipelining is to be avoided, more than one frame may be sent in a single transmission (i.e., without interrupting carrier between frames).

size and maximum frame size, and setting any two determines the third. Unfortunately, values that improve performance in the presence of noise conflict with those that minimize overhead. Therefore, it follows that there should be an optimal set of parameters for a given channel when operating at a given error rate and with a protocol that has a given header and ACK overhead.

This is indeed the case. A program[3] was written to compute analytically the overall expected efficiency of an AX.25 transmission, given the following parameters:

1. A Gaussian bit error rate (each bit error being independent of all others, as would occur with "white" thermal noise).
2. A transmission size in bits.
3. The number of frames into which to divide the transmission.
4. The overhead in bits of a frame header and an ACK.

For all bit error rates studied (10^{-2} to 10^{-7}, ranging from a link that is almost completely unusable to one so good that almost anything works), maximum overall efficiency was always obtained when each transmission was limited to one frame. As expected, the optimum transmission (and frame) size increases with decreasing bit error rate. The efficiency of a large (and suboptimal) transmission can be improved by dividing it up into several consecutive frames. However, this is always less than the efficiency attained when the message is divided up into smaller single-frame transmissions, even when the extra ACK overhead is taken into account. In other words, a "stop-and-wait" (i.e., window size one) protocol with an appropriate frame size always uses the channel more efficiently than a sliding window protocol with a window size greater than one.

The "real world," though, is a bit harder to characterize because many (if not most) errors are caused by collisions rather than insufficient S/N ratios. Errors are therefore much more likely to occur in bursts, with entire transmissions often rendered useless. Collisions are harder to model than thermal noise, so their effect on the efficiency of a sliding window protocol is harder to evaluate. However, it seems reasonable to argue that the "salvage value" of a multi-frame transmission that has encountered a collision is likely to be even less than one corrupted by thermal noise. This is because in CSMA, collisions tend to occur near the beginning of transmissions, during the "collision window" represented by the time needed for one round trip propagation delay. Since a hit in the first frame of a transmission renders all later frames unusable, dividing up the transmission into multiple frames simply increases header overhead without improving net performance.

Since we have shown that sliding windows are suboptimal for our application, and because they account for much of LAPB's complexity, the first of our motivations for scrapping LAPB altogether and designing a new protocol better tailored to the radio environment becomes apparent. However, we recommend that current AX.25 TNCs be operated in a stop-and-wait mode. Simply set MAXFRAME to 1 and take steps to adjust the packet size to a value appropriate for the channel. This ensures that new data can never collide with a returning ACK, regardless if the implementation already enforces a "single outstanding transmission" rule. One implementation strategy that might be useful here is to hold additional outgoing data in a buffer until any previously sent data has been acknowledged, and then send as much as possible in the next data packet subject to the maximum packet size limit. This is much more efficient than "pre-packetizing" outgoing data according to special characters in the data (e.g., carriage return), since short lines would otherwise result in small transmissions and poor throughput. A similar strategy, the Nagle Rule [10] has become widely accepted by implementors of the ARPA Transmission Control Protocol (TCP), the most common transport (level 4) protocol used in the ARPA Internet [6,7].

One final issue, that of "Selective Reject," becomes moot when LAPB is operated in a stop-and-wait mode. If only a single frame can be outstanding at any one time, there is no need for this special measures. In any event, simulations have: shown that Selective Reject actually *degrades* performance relative to "standard':' LAPB, especially on links with large window sizes, when errors occur in bursts (as they often do in the real

[3] See Appendix 1 for the derivation of the formulas used in this program.

world). [9] The same paper proposed an alternative known as "multi reject" but this is again unnecessary if the window size is one frame.

4. Congestion Collapse

Because LAPB made no provisions for channel sharing, as now implemented on amateur packet radio TNCs it is extremely prone to channel "congestion collapse," a stable condition where virtually every transmission results in a collision. Congestion collapse can be avoided only when TNCs become able to adapt themselves dynamically to the load offered by other TNCs. This section proposes congestion control algorithms that should be made part of the formal AX.25 protocol specification. They will be of little help unless widely implemented, because there is little incentive (other than altruism) for one to use them unless everyone does.

Several factors combine to cause our currently severe congestion problems. The first is *jump-on,* where several stations with traffic to send collide with each other the instant the channel becomes free. The second and more serious problem is caused by *hidden terminals.* These are stations unable to hear (and defer to) transmissions from certain other stations. When this happens, the CSMA (carrier sense multiple access) algorithm breaks down and collisions become very frequent. Kleinrock [3] has shown that just one or two hidden terminals in a packet radio network are often enough to degrade performance below that of slotted Aloha, and it doesn't take many additional hidden stations for performance to approach that of pure Aloha. Gower and Jubin [4] observe that carrier sensing actually makes things *worse* in the presence of hidden stations because it aggravates jump-on.

A more fundamental problem with packet radio is that all dynamic multiple-access schemes (regardless of carrier sensing, hidden terminals, delay and so forth) exhibit a "negative resistance" characteristic at high load levels. As the offered load increases, the collision rate increases and usable throughput decreases. While the exact point at which optimum throughput is obtained on a given channel varies widely depending on the algorithm and the station characteristics, there is always the need to control the offered load dynamically if it is to be operated efficiently. Unfortunately, this is not true for current AX.25 implementations.

4.1. Retransmission Backoff

The most important change that must be made to stabilize the AX.25 protocol is for successive retransmissions caused by the nonreceipt of an ACK to be spaced out further and further over time. T'his technique is called *backoff,* and many variations exist.

The most well-known example is *binary exponential backoff,* used in the Ethernet local area network. [5] In Ethernet it is possible to detect a collision during transmission. When this occurs, the transmission is aborted and a backoff timer is set to a random value distributed evenly between zero and a number that doubles after each unsuccessful attempt. In mathematical terms, th.e timer is set according to the expression

$$T = S\ random\ (0, 2^{min(n,10)})$$

where *random* returns a random number evenly distributed between its two arguments, n is the retry number, S is a proportionality constant, and the min function sets a maximum bound on the retransmission interval. This causes successive attempts by each station participating in a collision (there could be many stations all jamming each other simultaneously) to be spaced out over rapidly increasing intervals until the offered load on the channel drops to the point where successful transmissions can occur. As usual, should some retry limit (typically 16) be reached the packet is abandoned.

Base 2 gives reasonable performance and is easy to implement. However, other values can be used in the exponential calculation. Smaller bases back off less quickly, while larger bases cause rapid increases in the retransmission interval. If retransrnissions are more often caused by poor link margins than by collisions, a smaller base (i.e., slower backoff) may be appropriate at the expense of slower adaptation to congestion.

Packet radio differs from Ethernet in that there is no immediate indication of a collision; one can be detected only by the non-receipt of an ACK. Furthermore, the interval between transmission of a packet and the receipt of an ACK will vary depending on external factors such as channel speeds and the number of digi-

peaters. Therefore, more work is needed to adapt our backoff strategy to packet radio.

4.2. Retransmission Timing

In the most common implementations of AX.25 the estimated interval between transmission of a packet and reception of its ACK is called FRACK and is sent manually. An ad-hoc rule is used to increase it according to how many digipeaters are included in a connection.

There are several problems with this approach.

1. A constant value cannot adapt to changing channel conditions. A value that gives efficient and quick response when the channel is lightly loaded may cause many unnecessary retransmissions when ACKs are slow in returning because of channel load.
2. Few users bother to tune this value to an appropriate value. If anything, users are more likely to set it too small because they get anxious when their TNCs don't transmit when they "should"!

The presence of digipeaters is what makes this problem so difficult. There is no direct way to tell how much channel loading exists in sections of the network out of direct radio range, and hence no way to predict in advance how long one should wait before assuming that a transmission was lost in a collision. When digipeaters are phased out in favor of direct links between adjacent network nodes, a fixed interval with a timer that runs only when the channel appears to be clear should work well. In the meantime, however, we can borrow a technique from TCP and attack the static FRACK problern on long digipeater routes by computing it automatically. To do this we measure the time between the transmission of a given packet and the receipt of its ACK, always making sure that the retransmission timer is greater than this period. A "round trip timer" is created, providing continuous packet-by-packet measurements that take into account changing channel and digipeater loading.

Measured round trip times are only educated guesses about future behavior, because the path is statistical in nature (e.g., someone might unpredictably start a large file transfer through a remote digipeater). Note also that once the round trip timer is started, it should be allowed to run even if the packet being timed is lost and retransmitted. While this can cause anomalously large "spikes" in the round trip time measurements, the alternative (restarting the timer when retransmitting) is worse because an ACK for a previous transmission of the packet might come back immediately after the retransmission. This would yield an erroneously small estimate that might never be corrected. In any event, it is a good idea to process these measurements in two ways before using them to set the retransmission timer.

1. Compute a running average over several measurements to smooth out random fluctuations.
2. Include a "grace" factor to allow for sudden increases in the round trip delay.

The TCP specification recommends this formula for smoothed round trip time computation:

$$T_s' = \alpha T_s + (1-\alpha)\, T$$

where T_s' is the newly computed Smoothed Round Trip Time, T_s is its previous value, T is the round trip time encountered on an ACK just received, and α is a "tuning" constant ranging between 0 and 1. Large values of α cause T_s to react slowly to changes in measured round time, while small values cause it to respond more quickly. The TCP spec recommends an α value between 0.8 and 0.9. Dave Mills, W3HCF has shown [8] that using different values for α depending on whether the delay is increasing or decreasing has merit in the Internet environment. His experiments suggest using $\alpha = 3/4$ when $T > T_s$ and $\alpha = 15/16$ when $T < T_s$ to give a "fast attack" and "slow decay" response characteristic.

Once T_s has been computed, TCP specifies that it be multiplied by another tuning constant, β, to arrive at the final retransmission timer value. The recommended value for β is 2.0, i.e., the transmitter waits two round trip times for an ACK before retransmitting. The TCP specification also recommends that repeated unsuccessful transmissions be spaced out, although the exact backoff algorithm is not specified.

For AX.25 use, we can combine the round trip timer from TCP with the binary exponential backoff algorithm from Ethernet

and use them together to set the retransmission timer. To do this, we first observe that we should always wait at least one round trip interval (preferably β intervals, to allow a grace interval as in TCP) for an ACK to come back. One possible result is the following:

$$T = T_s \beta \, random(1,2^n)$$

When packets are not being lost, the round trip timer will always be set to β times the smoothed round trip interval. Should several retransmissions occur in a row, however, both the average retransmission interval and variance will double on each retry.

4.3. Persistence

The round trip timing and backoff algorithms just described help stabilize the channel and prevent congestion collapse through negative feedback that attempts to operate the channel near the peak of its throughput efficiency curve. These are especially helpful when many hidden terminals are present, limiting the maximum attainable channel efficiency. Even without hidden terminals, however, efficiency can still suffer greatly in a heavily loaded network because of the jump-on problem.

One way to alleviate this is to change how stations with packets to send behave when they find an idle channel. Our current algorithm, namely "transmit as soon as you hear the channel go clear," is formally called *1-persistent CSMA* and leads to the jump-on problem. A less greedy alternative is *p-persistent CSMA,* where each station waits a random amount of time before transmitting even when the channel seems to be clear. Each station generates an evenly distributed random number between 0 and 1 and compares it to a constant, *p,* which also ranges between 0 and 1. If the random number is less than *p,* the station transmits; otherwise it waits a small amount of time (called the *slot time,* ideally one round-trip propagation delay) and repeats the procedure. As *p* approaches zero, channel efficiency theoretically approaches 100%. Unfortunately, however, delay also approaches infinity! For any non-zero value of *p,* however, the packet will eventually be sent after finite delay although channel efficiency will decrease.

Persistence works because the stations waiting for the channel will "spread out" their transmission attempts once the channel goes clear. Assuming only one station decides to transmit in each slot, or round trip interval, the other stations on the channel will hear and defer to it before they attempt to transmit themselves.

Clearly there is an optimum value of *p* for a given level of channel activity. If *p* is too large, too many stations will jump on each other in the first slot. If *p* is too small, many slots will go to waste before the stations eventually transmit, and transmission delays will increase unnecessarily. It is therefore best to set *p* dynamically, if possible. At low load levels, $p=1$ gives the best performance; as the channel reaches 75-80% offered load (i.e., the channel appears busy 75-80% of the time with either good packets or collisions) the value should be decreased.

The backoff algorithm given in the previous section can be viewed as a way to adjust the value of *p* dynamically when timeouts occur, since doubling the retransmission interval corresponds to halving the value of *p.* The only real difference is that the backoff interval is evenly distributed while the persistence "timer" is exponentially distributed.

TAPR TNCs pioneered the use of DWAIT, a persistence-like feature. It should be enhanced to provide true persistence (i.e., by randomization). A worthwhile research project would be the development of a good algorithm for the real-time evaluation of *p*, with the goal of incorporating it into a future revision of the AX.25 specification. One possible approach is to sample the state of the DCD line periodically and estimate the channel activity. The value of *p* could then be found in a lookup table. If more retransmissions occur than expected for the measured amount of traffic (e.g., due to hidden terminals), then *p* could modified as appropriate.

5. Some Final Thoughts on Congestion Control

Congestion control in packet radio networks and packet switching networks in general are notoriously difficult problems and have received much research attention. The suggestions made here, however, should greatly alleviate the problem and make our AX.25 digipeater networks halfway usable under heavy load.

However, it may turn out that radically new approaches to transmitting bits and packets over amateur radio links will hold a better solution.

1. Spread spectrum with different spreading sequences assigned to each receiver eliminates channel collisions except for the less likely case where two transmitters attempt to send to the same receiver at the same time.
2. Since the received signal levels in a packet radio network usually vary widely, modulation methods and forward-error-correcting (FEC) techniques that exhibit higher degrees of "capture" may help by minimizing collisions where nobody "wins."
3. Busy Tone Multiple Access (BTMA) [3] involves receivers indicating on a separate radio channel (with a "busy tone") that they are actively receiving a packet. Stations monitor the busy tone rather than the data channel to determine when to defer. If the power levels and propagation conditions between the main and busy-tone (channels are symmetric, it is possible with BTMA to avoid completely the hidden terminal problem. While this technique requires twice as many radios (along with the ability to operate in full duplex, although this could be crossband) the improvement in throughput could easily more than double.
4. Conventional "bent-pipe" analog or regenerative digital repeaters of the split-frequency type also reveal hidden terminals. While this is contrary to the trend toward digipeaters, they may be the most expedient way to solve a particularly difficult case.

This list only touches the list of possible approaches to congestion control. Above all, we need to leave plenty of room for experimentation into these and other "low level" problems. Our experience has shown that there *cannot* be a single, "standard" link level protocol that is optimum everywhere. Each link protocol must be customized to a specific environment, and the network architecture (e.g., higher level protocols) must take this fact of life into account and be flexible enough to accommodate them all.

6. A Replacement for LAPB

As we have seen, LAPB is out of its natural element (a reliable, full duplex point-to-point link) when it is used on a lossy, half duplex packet radio channel. LAPB lacks the features needed to control congestion and improve performance on bad links. It also contains unnecessary "features" that increase the size of an implementation and may actually degrade performance. This section is an attempt to apply the lessons learned from LAPB to the design of a new protocol, one much better suited to the amateur radio environment.

6.1. Link Level Acks versus Connections

An important point needs to be stressed here. Link level *acknowledgements* (desirable for performance reasons when operating on a lossy channel) do *not* necessarily imply link level *connections.* A packet switch operating in a large metropolitan area may serve a total of several hundred users, but only a small fraction may be active at any one time. Why require the switch to maintain hundreds of mostly idle link level protocol control blocks for all these users, or alternatively saddle users with the burden of setting up a link level "connection" to their local switch before they may communicate with their final destinations'!

Veterans of the amateur packet radio "protocol wars" will recognize this as an argument for datagram-oriented networks. We believe that the connection- (or virtual-circuit-) oriented nature of LAPB is another major contributor to its complexity. We further believe that a redesigned protocol that enhances link level reliability with acknowledgements *without* explicit connection management procedures will be both simpler and easier to implement.

6.2. Window Control

As shown by our analysis of LAPB, sliding windows are unnecessary in a half duplex packet radio environment, so our replacement is a simple stop-and-wait protocol. Even when traffic is pending to more than one station, only one data packet may be sent at a time. An ACK must be received for this packet (or a "give up" interval exceeded) before another packet can be sent to this or any other station. This avoids the ACK collisions that could occur if we were to sent traffic to two or more

stations at one time. This also simplifies the implementation, since all traffic to be sent on a given channel can be kept on a single queue regardless of destination.

6.3. Acknowledgement Piggyback

Another feature of LAPB that makes its contribution to complexity is the ability to "piggyback" an ACK on a data frame traveling in the reverse direction. This is a difficult feature to use in practice, because there is seldom a data frame available at precisely the right time on which to piggyback an ACK. Most applications operate in a "pseudo half duplex" mode: one host sends one or more packets to its peer, and one or more packets later flow in the opposite direction. Some protocol implementations delay the transmission of ACKs in the hope that a higher layer will soon generate an outgoing data frame on which it could be piggybacked. This seldom succeeds, however, because any delay must be kept short both to prevent unnecessary retransmission and to keep the round trip time small enough to avoid affecting throughput. We therefore felt that ACK piggybacking was not necessary in our protocol.

6.4. Acknowledgment Retransmission

In LAPB, a lost ACK causes the sender to timeout and retransmit a data frame, even though it has already been received correctly. When loss rates are low, this is not a serious problem. However, should a substantial fraction of the ACKs be lost, the unnecessary retransmission of data frames by the sender merely to elicit ACK retransmissions from the receiver can cause a considerable loss of performance. This problem was first recognized eleven years ago by the designers of the first packet radio network, the ALOHANET, who proposed an elegant solution that we have dubbed the ACK-ACK protocol[4] [11].

As its name implies, the ACK-ACK protocol provides for the acknowledgement and retransmission, if necessary, of ACKs to prevent the unnecessary retransmission of data packets. Let's look at an example using LAPB more closely to show why ACK-ACK gives better performance. Assume that the probability of a data packet or ACK being successfully received in one try is p_d or p_a, respectively. In order for the sender to expect to receive one ACK, the receiver will, on the average, have to transmit its ACK $\frac{1}{p_a}$ times. In order for the receiver to transmit this many ACKs, however, it will also have to receive $\frac{1}{p_a}$ data packets. Therefore, the sender can expect to send $\frac{1}{p_d} \times \frac{1}{p_a}$ packets for each ACK received. For example, if the probability of a data packet or ACK being successfully received is 25%, then $p_d = p_a = 0.25$, and the sender will have to send each data packet an average of sixteen times before an ACK is finally received and it can continue with the next data packet! On such a link, a greater attempt by the receiver to ensure receipt of its ACKs by the sender is worthwhile.

Given that nothing can be done about the "raw" value of p_a, then the only thing the receiver can do to increase reliability is to retransmit ACKs whenever necessary. If the receiver were to retransmit each ACK up to N times, then the probability that at least one attempt succeeds is

$$1 - (1-p_a)^N$$

If the probability that a data packet is successfully received is p_d, then the expected number of data packet transmissions that result when the receiver makes N attempts to transmit each ACK is

$$\frac{1}{p_d} \times \frac{1}{1-(1-p_a)^N}$$

For our earlier example where $p_d = p_a = 0.25$, if $N=5$ then the probability becomes .76 that at least one ACK will make it back to the sender. This decreases the expected number of data packet transmissions from 16 to 5.25, a substantial improvement.

The receiver does this by setting a timer when it transmits its ACK and retransmitting it if an acknowledgement of its acknowledgement (an "ACK-ACK") does not return before the timer expires. The receiver may also accept the next data packet from the sender as confirmation that its ACK of the preceding one was accepted, because the sender will not continue with the next data packet until the one

[4] We considered calling this the "Bill the Cat Protocol," but thought the reference too obscure.

outstanding has been acknowledged (remember this protocol operates in stop-and-wait mode). This means that when one station sends a steady stream of traffic to another, no additional packets over the conventional protocol case are sent. Only at the end of a burst of traffic (or after an isolated packet) is an extra ACK-ACK packet generated. Clearly, the ACK retransmission timer must be shorter than the data retransmission timer; this ratio corresponds to the value of N in the above equations, the number of times the receiver will attempt to retransmit the ACK before the data packet is retransmitted unnecessarily.

To make the protocol reliable, we need to include a *frame identifier* (ID) with each transmitted 'data frame and ACK. The ID ensures that the sender and receiver do not get out of step, possibly losing or duplicating a data frame. The ID need not be a sequential value; it need only be different from one frame to the next. If the receiver receives a data frame with the same source address and ID as the last received frame, a normal ACK is sent but the frame is otherwise ignored as a duplicate.

The ACK-ACK protocol can be viewed as establishing an implicit, unidirectional "connection" with the first data frame, which exists only as long as there is data to send. Once an ACK-ACK is sent to show that no more data is available, the sender and receiver need retain no further state (i.e., addresses and ID) and the "connection" is torn down. The sender then repeats the: process with any other station for which it has traffic.

Only one implicit "connection" exists at any moment (although the destination to which we are "connected" can change very rapidly) since our half duplex operating rules require that only one data packet be outstanding at a time. If a third station sends us data while we're already exchanging data with another station, then this new data is put on a queue and processed after we've completed our current transfer sequence. If a data frame requesting an ACK should arrive from a new station, we do not immediately acknowledge it because that would encourage it to send more data. This might interfere with the station we're already communicating with, so we hold this frame on a queue for processing once we return to an idle state. (However, an ACK from such a station may be processed immediately, as may incoming data that does not request an ACK). There: is of course the chance that we might delay processing of the new data so long that a tirneout occurs. This can be largely avoided, however, if the link retransmission timers are chosen to be longer than the time needed to send a typical multi-packet "burst." These can be limited by appropriate window sizes in the end-to-end (transport) protocol. The alternative, immediately acknowledging the new data but telling the sender to hold off on more, is possible but complicates the protocol. Periodic "probes" from the other station would be needed to guard against the deadlock that would occur if our "go ahead" message is lost, and these could also collide with packets to or from the station we're already communicating with. The simple acknowledgement delay strategy thus seems workable, especially if the data retransmission strategy backs off rapidly (e.g., exponentially). It also simplifies the code greatly, since managing multiple connection state control blocks is usually the hardest part of a "multi connect" TNC.

Since radio channels can vary widely in quality, it is desirable to make the use of ACK-ACKs and even ordinary ACKs optional. This is especially useful when supporting datagram-oriented network layer protocols such as IP, since it is more efficient to dispense entirely with the overhead of link level ACKs and rely on end-to-end retransmission of lost packets when link error rates are very low. Our protocol therefore provides for an indication in each packet (data or ACK) whether a reply (ACK or ACK-ACK) is expected for this transmission. It is therefore easy to adapt this protocol to changing radio conditions or user reliability requirements. An ACK-ACK need be sent only when no more data is available, since the next data packet awaiting transmission may serve as an ACK-ACK. An ACK-ACK may then be represented merely by an empty data packet with the "acknowledgement requested" bit turned off. This simplifies both the concept and the implementation of the protocol considerably.

6.5. ACK-ACK Preliminary Specification

In specifying frame formats corresponding to data, ACK and ACK-ACK messages, we have tried to adhere to the basic structure of the AX.25 datagram sublayer. In other words, we keep the standard AX.25 address field lay-

outs and attempt to use the existing control flag definitions whenever possible. Data is transmitted as a UI frame with poll and command turned on, ACK is transmitted as a UA frame with final and response turned on, and ACK-ACK is transmitted as an empty (no data in the data field) UI frame with poll turned off and command turned on.

6.5.1. Frame Formats

The frame format is similar to that of AX.25. The fields are named and the length of the field in bits follows the field name and is enclosed in parenthesis. Here is the basic frame:

F (8)	Dest (56)	Src (56)	Digi *(O-448)*	Ctl (8)	PID (8)	ID (8)	Data	FCS (16)	F (8)

Where

F Flag, length *8* bits. Value 7E hex, not bit stuffed.

Dest Address of destination, length 56 bits. This is the standard AX.25 address/SSID combination.

Src Address of source, length 56 bits. This is the standard AX.25 address/SSID combination.

Digi Address of repeater(s), length varies from 0 (no digipeaters) to 448 (8 digipeaters). This is carried over from AX.25, although deprecated for the networks in which this protocol is likely to be used.

Ctl Control field, length *8* bits. The content determines the frame type, either UI or UA. The following values are in binary:

UI 000p0011

U A 011p0011

where 'p' is the poll/final bit.

PID Protocol identifier, length 8 bits. This identifies the layer three protocol and follows the same conventions as AX.25.

ID Frame ID, length 8 bits. This uniquely identifies the frame from the previous and subsequent frames to avoid duplicate and lost frames.

Data Data, length 0 to 523,688 bits (0 to 65,461 bytes). The sender and receiver should agree on a maximum frame length not to exceed 65,461 bytes in length.

FCS Frame check sequence, length 16 bits. This contains the CRC-CCITT checksum of the frame.

F Trailing flag.

The ID byte is probably best set from a single counter used for all transmissions regardless of destination. The only requirement placed on the sender is that the value of the ID field not be duplicated in any two subsequent frames sent to the same destination; this can be avoided by limiting the transmit queue to 255 entries.

6.5.2. Flow of Data Between Stations

This section is a preliminary specification of the ACK-ACK protocol in an informal, narrative style. As this definition is refined and implemented, a more formal description is being written. Contact the authors for more information.

Data is transferred from station A to B in the following way. Station A selects a ID value, sends the data (UI) frame to B, and starts a timer with interval T_d. B notes the sender address and ID number, responds with an ACK (UA) containing the ID just received, and starts a timer with interval T_a. After receiving an ACK. from B for the most recently sent frame, A responds by sending the next data frame and restarting its timer if there is more data to send, or sending an ACK-ACK (empty data frame without poll request) and stopping its timer if there is no more data to send. When B receives the next data frame or an ACK-ACK, B discards any ID it is currently retaining from A and either sends an ACK with the new ID (restarting its timer) or goes into the idle state (stopping its timer) as appropriate.

If A should fail to receive an ACK containing the correct ID from B within the timeout interval T_d, then it *increments* a retry counter, retransmits the: data frame and restarts its timer. If the retry counter exceeds a fixed limit, additional transmission attempts for this frame are aborted, and if possible, the packet should be returned to its original source with an error indication. Higher level protocols are then responsible for any further error recovery procedures. If B fails to receive either another data frame or an ACK-ACK from A within its timeout period T_a, then it resends its ACK,

restarts its timer and increments a retry counter. If the same data frame is received again from A, B resets its retry counter, resends an ACK and restarts its timer but otherwise ignores it as a duplicate. If there is no response at all from A after N ACK retransmissions, where N is the ratio $\frac{T_d}{T_a}$, B abandons any further retransmission at tempts and acts as though an ACK-ACK had been received (i.e., it discards the ID and sender information and returns to a quiescent state). Higher level protocols are responsible for detecting and recovering from the residual possibility for packet duplication this allows.

Because of its small amount of state, this protocol can be implemented simply and with little memory. Communicating with several different stations in "rapid fire" sequence is easy; the receiver need retain the last ID valued received from a particular sender only as long as data is actively being transferred. Special connection establishment packets are unnecessary.

7. Selection of Appropriate Values for Tuning Parameters

There are three parameters that should be regularly adjusted or "tuned" to the existing conditions on the channel. These parameters are frame size, transmit timeout timer T_d, and ACK timeout timer T_a. The frame size should be adjusted to the optimum value based on the current channel statistics. As more frames are lost the frame size should be shortened until the optimum operating point is reached as described in Appendix 1.

The correct timer values are much simpler to calculate. The transmit timer should be set to some value that is significantly greater than the round-trip time for a frame to travel from sender to receiver for an ACK to return. The receiver should also calculate the round-trip time and determine how many ACKs are required to insure that an ACK reaches the sender. The receiver should set the value of T_a to be T_d divided by the number of ACK transmissions N necessary to increase the probability that an ACK arrives at the sender to the desired level. In order for this technique to work properly both the sender and receiver need to agree on the formula that relates round trip time to the selected value of T_d. Both timers should be randomized and backed off as described earlier in the congestion control section.

As soon as T_d (equal to NT_a) expires at the receiver the receiver should cease sending ACKs. This prevent the ACKs from colliding with and destroying a retransmitted data frame. If the sender should fail to receive an ACK it will resend the frame. The receiver repeats the process of acknowledgement but discards the frame.

7.1. Performance

ACK-ACK was compared to AX.25 for various links and message sizes using a Monte Carlo program that simulates both ACK-ACK and AX.25 in identical environments. In all configurations a window size of 1 (stop and wait) was found to be most efficient and ACK-ACK to be more efficient than LAPB/AX.25. Performance improvements over LAPB/AX.25 ranged from a low of 1.21 for very reliable links to a situation involving 2 digipeaters where ACK-ACK delivered the data and AX.25 failed to deliver all the data within the running time of the simulation. In the latter case, when the simulation was terminated the performance improvement over AX.25 was already over 100.

7.2. Conclusion

LAPB/AX.25 is a good protocol for point-to-point communication over reliable full-duplex links. ACK-ACK is more desirable than AX.25 in high error rate and/or half-duplex environments because it provides significantly greater throughput, better channel utilization, and is much simpler to implement.

8. References

[1] Fox, T., ed "AX.25 Amateur Radio Link Layer Protocol Version 2'', American Radio Relay League, 1985.

[2] CCITT Study Group VII, "Interface Between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) For Terminals Operating in the Packet Mode on Public Data Networks," Recommendation *X.25, 1984.*

[3] Kleinrock, L., and Tobagi, F., "Random Access Techniques for Data Transmission over Packet-Switched Radio Channels," National Computer Conference, 1975. pp187-201.

[4] Gower, N., and Jubin, J., "Congestion Control Using Pacing in a Packet Radio Network," IEEE Conference on Military Communications, 1982.

[5] Metcalfe, R. M., and Boggs, D. R., "Ethernet: Distributed Packet Switching for Local Computer Networks," Commun. ACM, vol. 19, pp. 395-404, July 1976.

[6] Postel, J., ed., "Transmission Control Protocol Specification," ARPA RFC 793, September 1981.

[7] US Department of Defense, "Military Standard Transmission Control Protocol," MIL-STD-1778, 12 August 1983.

[8] Mills, D., "Internet Delay Experiments," ARPA RFC 889.

[9] Brady, Paul T., "Performance Evaluation of Multi-Reject and Selective Reject Link-Level Protocol Enhancements," Bell Communications Research. (Paper submitted to ICC Toronto, June 1986).

[10] Nagle, J., "Congestion Control in IP/TCP Internetworks," ARPA RFC 896

[1 1] Binder, R., et al., "ALOHA Packet Broadcasting - A Retrospect," Proceedings of the 1975 National Computer Conference, page 208-209.

Appendix 1: Efficiency of a Go-Back-N Protocol

Two interrelated factors must be taken into consideration when evaluating the efficiency of a go-back-N sliding window protocol on a noisy channel: the effect of header overhead (limiting efficiency even on an error-free channel) and the effect of garbled frames. This section derives the formulas for these two factors that were used to arrive at the conclusion that a window size of 1 (i.e., a stop-and-wait protocol) maximizes channel efficiency when operated on a half duplex channel. To determine the net efficiency of the channel, these two factors are evaluated with the desired parameter values and then multiplied.

Header Overhead

If there are D data bits in a transmission, H frame header bits, and N frames in a transmission, then the ratio of data bits to total bits sent is

$$\frac{D}{D+NH}$$

If we include the overhead of an ACK (A bits) then this becomes

$$\frac{D}{D+NH+A}$$

Unusable Frames

If the channel bit errors occur independently at rate R, (i.e., caused by Gaussian sources such as "white" thermal noise) then the probability that a frame of length $D+H$ is received without errors is simply $(1-R)^{D+H}$. In this section we call this quantity G, the probability of receiving a good frame. Note that G decreases as we increase D; i.e., the longer the frame, the greater the chance it will be corrupted in transmission, unless the channel has a perfect bit error rate ($R=0$).

In the go-back-N error recovery technique, only the next expected frame can be processed; any other frames received are discarded, even if they are received correctly (i.e., with a valid CRC).

Consider this example. Four frames numbered 1, 2, 3 and 4 are sent in one transmission. Frame 1 is received normally, but frame number 2 is corrupted. Even if frames 3 and 4 are received correctly, the receiver will only acknowledge frame number 1. Frames 2, 3 and 4 will have to be resent, even though only frame 2 was actually lost.

To analyze the performance of a sliding-window protocol with go-back-N recovery, we need to find the expected number of *usable* (with correct CRC and in correct sequence) frames in a transmission containing N frames. We then divide this number by N, yielding the normalized efficiency of the protocol (the number of usable frames received divided by the total number sent). G is defined as the probability of any specific frame within the transmission being received correctly, and we assume that this value is the same for all frames (i.e., the frames are all the same size and errors are independent). Clearly, if we

were able to use every correctly received frame (i.e., if we did not discard frames received out-of-sequence due to the loss of an earlier frame) then this efficiency would simply be G, for any value of N. In the go-back-N case, however, the analysis is more complicated.

The probability that zero frames are usable is (l-G), i.e., the probability that the first frame is in error, rendering it and all later frames unusable. The probability that only one frame is usable is G(l-G), the probability that the first frame is received correctly and the second one in error, and so on up to the probability that all N frames are usable. Summing over all possibilities and weighting by the: number of usable frames for each, we obtain

$$(1-G)\sum_{i=0}^{N-1} iG^i + NG^N$$

By factoring out G from the series and then integrating and differentiating the finite sum, we obtain

$$(1\text{-}G)\ G\frac{d}{dG}\int\sum_{i=0}^{N-1} iG^{i-1}\,dG + NG^N$$

Integrating,

$$(1-G)\,G\frac{d}{dG}\sum_{i=0}^{N-1} G^i + NG^N$$

This now contains the sum of a finite geometric series. Substituting with the closed form of the sum, we obtain

$$(1-G)\,G\frac{d}{dG}\frac{G^N-1}{G-1} + NG^N$$

Differentiating,

$$(1-G)\,G\left[\frac{1-G^N}{(1-G)^2}-\frac{NG^{N-1}}{1\ -\ \mathrm{G}}\right] + NG^N$$

Simplifying,

$$G\frac{1-G^N}{1-G}$$

And normalizing by N, the number of frames sent in each transmission, we finally obtain

$$G\frac{1-G^N}{N(1-G)}$$

A quick check shows that substituting N=l for the single-frame case yields G, as expected.

PERFORMANCE ENHANCEMENTS FOR THE AMATEUR PACKET NETWORK

J. Gordon Beattie, Jr., N2DSY
Thomas A. Moulton, W2VY

The Radio Amateur Telecommunications Society
206 North Vivyen Street
Bergenfield, NJ 07621
201-387-8896

ABSTRACT

For several years the amateur packet network has been using a link level networking system based on "digipeaters". This system has allowed the user community to expand rapdily. As the packet mode has increased in popularity, many problems have developed that have caused the users some inconvenience. It is the feeling of the authors that the degraded performance levels found on the amateur packet network are largely due to the simplistic digipeater network approach.

This system has also forced users to provide control mechanisms in relative isolation. This has had the effect of reducing reliability and performance. An X.25 level 3 packet switch can provide improvements in the areas of retransmission, routing, addressing, quality of service negotiation, access mangement and data flow control.

We will examine these areas, showing how a small but intelligent packet switch can improve user operations and network performance. While discussing the enhancement of switching facilities we will also explore some of the user features and facilities available or planned for the network.

RETRANSMISSION

Digipeaters do not have any capability to retransmit frames lost in transit. If a frame has to traverse four hops to get from the sender to the receiver and on the third hop it is lost, the sender must retransmit the frame on the first hop again. If the frame traverses all the hops without getting lost, then the receiver will send an acknowledgement. This situation is degraded by the fact that the acknowledgement must also get through unmolested. If either fails, retransmission must occur.

This retransmission timeout must be fairly long so that the frame and its acknowledgement can get across the network. On long links this could be a significant period of waiting time. If the acknowledgment is lost, we can have both the retransmitted data and the acknowledgement frames causing additional network congestion.

This situation is similar to a student who must successfully pass grades 1-4 without failure. In the event of a failure, the student is not allowed to go back and repeat the one grade missed. Instead, the student must return to the first grade and restart the academic progression. Under such a system, we would have many first and second grade students and an incredible dropout rate !

In the packet network our "dropouts" are connections that have had an excessive number of retransmissions and have, as a result, failed.

The packet switch will provide hop-by-hop acknowledgements. This effectively will eliminate most cases of end-to-end retransmissions.

ROUTING/ADDRESSING

The present digipeater system has a limited routing capability which allows a user to specify a set of up to eight digipeaters for his connection path. This field is carried in every frame. It not only limits network connectivity, but presents a large overhead to the network.

This path may or may not be ideal for the service required by the user, but instead reflects the individual's "best guess" on path availability and performance.

Further, the user has little information to help decide whether the establishment of this new connection will have an adverse effect on existing connections. The end result is that there is a tendency to "pile on" already congested network channels and machines.

The preferred method would allow the user to specify the desired destination user and location, leaving the routing decision to the network itself. This

approach may seem unfair because the network has pre-empted the users' ability to set his own routing. This approach is FAIRER because a single user can't unilaterally and inadvertently (hopefully !) cause network connection failures.

The use of an addressing scheme that is more stable than the amateur callsign would allow implicit routing. Systems based on the CCITT X.121 and either the telephone system or grid squares have been suggested. Such schemes would be used in conjunction with the callsign to provide a network connection. The user connection request format could be:

"C N2WX @ 03100305" or
"C W2VY @ 03100201596~

Such information would be adaquate for a network to locate the specific network, country, region, LAN and user. The format can be as specific as desired. If there are several local networks in place, each with many users, the routing task would be simplified by the provision of more detailed addressing as shown in the second example.

QUALITY OF SERVICE

Under the present system it is assumed that the user can serve his own needs best by allowing him to select the internal network path and parameters. This user decision is made with NO performance information except that "it seems to work for ME". This lack of information has often caused congestion at critical network points. This congestion has led to user dissatisfaction and frustration.

In order to be good neighbors, we need to use the network cooperatively. This cooperation can be administered by the packet switch. The switch can receive requests for certain performance characteristics and "negotiate" them with the requesting station. This will allow an emergency user higher priority than normal traffic, while also providing different performance characteristics to different connection types. These connection types would include:

1. Terminal to terminal
2. Multiterminal (roundtable)
3. Terminal to machine
4. File transfer
5. Mail forwarding
6. Voice transmission
7. Image transmission

Each of these connection types has differing requirements for data volume, delay, and accuracy. Parameters can be grouped to provide a Quality Of Service profile required to support a given connection type.

The switches will have the capability of building a "service" virtual circuit between adjacent nodes to regulate the behavior of the network. Ultimately, the network will be able to transfer connections to new paths.

CHANNEL ACCESS AND DATA FLOW CONTROL

The intelligent packet switch will also provide channel access control. This scheme allows the network to decide when a particular user may send another frame. This control will be exerted through a system of timers. Separate timers will be used to regulate the flow of expected data, acknowledgments and unsolicited frames. This scheme will not prohibit unsolicited transmissions, but will regulate the flow according to overall channel activity and connection type. The basic philosophy will be for acknowledgments at link and packet level to be sent with less delay than "data". The plan also calls for the switch to use shorter timers than "users".

Other ideas center around the idea of allowing the user to transmit until the transmission window is closed. Then leaving the window closed until the switch has serviced other users on the channel. Such changes in operation will not require changes in the AX.25 protocol, but simply cooperation in the setting of timers and counters,

UPPER LAYER SUPPORT

The users of this network service will need to experiment more with different session types. Through this experimentation, we will be able to develop specific operational parameters for them. The network proposed here will use the CCITT X.25 packet level protocol over the amateur AX.25 link level, This will allow us to have a reliable "data-pipe" for the great number of amateurs who will require such service.

It will also give us a strong backbone capability from which lower classes of service can be derived. For example, packet voice users may not require, nor desire, retransmission. It will however require that delay be minimized. If a session is identified as a "voice" session the network connection request will reflect these requirements.

Because we have developed a strong reliable system, the user is not obliged to provide a complicated transport protocol for routine terminal-oriented connections. Use of CCITT X.224 Class I would provide protection against any residual connection or packet loss. This protocol is not complex and as such could be implemented on small terminals and computers. It uses three bytes of overhead in data transfer. This is important when you consider that many links are still slow speed.

SUMMARY

Based on our professional and amateur experience we felt that the development of a small X.25 packet switch would be possible. Howie Goldstein, N2WX had written the TAPR TNC-2 software and was interested in providing a "real" networking capability. Our joint efforts have produced an initial set of level 3 features needed to support terminal-to-terminal and file transfer communications through a network. The results of Howie's coding have produced an integrated X.25 PAD (Packet Assembler/Disassembler) and Switch for the TNC-2 and Xerox 820.

The members of the Society strongly urge that present digipeater sites are converted to virtual circuit-based packet switch sites. Since Xerox 820s and TNC-2s are often found in the role of a digipeater, it is not a major problem to change the software thereby upgrading the digipeater to a switch.

CONCLUSION

We feel that our experience with the basic level 3 software for the TNC-2 has allowed us an unique opportunity to experiment with a true level 3 networking environment. We hope that others interested in bringing up X.25 level 3 networks will contact us with their observations and recommendations.

THE **PACKETMASTER PACKET SYSTEM** FOR **CP/M AND DOS** COMPUTERS

Bernie Mans, AA4CG
PakTek Packet Technology
P.O. Box 3010
Crystal River, FL 32629

Abstract

This paper outlines the **PacketMaster** system, **a flexible packet program and simple peripheral hardware** to add AX.25 to the repertoire of the personal computer. N.B.: This is much more than a trivial variation on one of the many terminal emulation programs commonly used with external self-contained TNCs. I describe here a true packet peripheral intimately connected with a host computer unlike the typical terminal emulator/tnc combo. In the generic tnc, the tnc is only loosely coupled to the computer (or terminal) usually via a simple RS-232 1200 baud serial line. **The advantages of the peripheral approach over the serial method are the enumerated later in this paper.**

Observations on
First Generation Packet Systems

There has been a great emphasis on the development, manufacture, and use in the amateur community of what are essentially black box appliances--elegantly packaged microcomputers with firmware documented at only the most basic user level. This was acceptable in the early stage of the development of amateur packet radio (only one or two years ago!) when:

a)Users were unsophisticated so just about anything that performed at all was considered adequate.

b)There was minimal freedom of choice: there were only two or three tncs to choose from.

c)There were only hundreds of users dispersed across the United States. So there were no great concentrations of activity anywhere and there were large segments of the country where packet activity was virtually nill.

Second Generation Packet Systems

I propose a second generation of **virtual tncs.** The unifying characteristic of these advanced tncs is that packet capability will be incorporated into true general purpose (gp) computers by simple extensions to their hardware. The major advantage of this integration is that the power of the gp computer can be applied to either the execution of an unmodified packet program or to the permanent modification of it. Not only can extended packet functions be easily accomodated by the host computer but the packet system itself may be readily customized utilizing the resources of the same computer.

The capabilities of current first generation tncs will be extended in second generation tncs. These extensions will be discussed next.

DISPLAY CHARACTERISTICS--

a)The 80 character x 24 line display will become the standard format.

b)Split-screen scrolled operation will be the norm with two independent, approximately equally sized windows for received and transmitted packets, respectively.

c)There should be a command area near the bottom of the screen when in the command mode.

d)There should be a status line at the bottom of the screen to display the time, the current link status, and any other interesting transient activity.

e)If multiple connects are initiated, the windows should automatically subdivide to display maximal information.

f)Screen updates and scrolling will be fast, i.e. they will not be limited by the baud rate of a serial line.

COMMANDS--

a)The command set should be extended and simplified for operation with video displays. The WA8DED firmware for the TNC-1 is a excellent illustration of

what cart be done in th is doma in.

b) The tnc should be optionally under complete computer control. Connects should be accepted and appropriate data logged to disk. There should be an extended, integrated mailbox and message facility. Operational characteristics could be varied and optimized for a particular application by editing a simple command file.

c) There should be onl ine help available at a user selectable level of completeness. i-eve 1 Ø cou ld be expert level with minimal prompt ing; level 3 coulld be defi ned as novi ce level with the quant i ty of help given/available correspondi rigly greater.

PERFORMANCE--

a) The system shoulld utilize a high per-f ormance modems that ut i l i ze stable internal crystal controlled clock generators. A good choice i s the Am79 i0 farni ly of modems. These LSI devices do not not requi r-e either initia3 or period ic ca 1 i brat i cm.

b) Even more exci t i ng l y, these modems may be switched from 1200 baud to 300 baud operation under computer contro1 or manually via a simple toggle switch!

c) Anot her advantage of t h e 7910 family of modems i s that they per f orm well without the necessity of external act ive analog or digital filtering.

USER EXTENSION AND MODIFICATION--

a) Because the appl icat ion code is not effectively hiddert in EPROM arid segments of comment ed source code wi l 1 be made available, the moderately sophist icated user may customize the user interface at will. This is important when a user wishes to handle messages in a specific predef i ned format (such a s an NTS messages). TCP/IP network functions could be included as a true amateur network evolves. Ot her eriv i roments could be created for d i f ferent purposes.

b) The system interface (BIOS) code st i 11 isolates the user from the vagaries of the hardware operation. This we1 l-document ed f irmware should enable the programmer without art electrical engineering background to effectively utilize the packet interface.

Background

The initial goal of this des i gn/development proj ect was to provide a simple, i nexpens i ve means to adapt my Kaypro 10 comput er syst em to packet rad io. The Kaypro is a moderat e performance packaged CP/M system ut i l i zing early-80's technology. It is the latter of a f i ne l i ne of computers start ing with the Ferguson Bi g Board and the Xerox 820 (of p a c k e t rad io fame!), As such, it5 CPU is the Z80 microprocessor arid i t s bus is compat i ble with the Z80 family of peripheral chips. Most import ant 1 y, it contains the Z80 SIO chip. The onboard SIO, when properly init ial ized, is capable of SDLC (i.e., packet) operation with a minimum of external circuitry (the Z80 and SIO combo form the basis of the TAPR TNC2 and i t s many cl ones for those of you who still have doubts about its packet capabi 1 ity).

Why d idn' t I j ust purchase a TNC2 and use the Kaypro as art intelligent termina3? Because!:

a) The TNC2 or one of its many Cl ones was rut yet available when this project was started .

b) I felt (and still feel) that the use of a "real" comput er as a terminal emul at or i s a waste of resources. This is evert more true when the host comput er contains a Z80 and Z80-SIO, the basis of the TNC2. I ask, "Why not j ust buy or build a terminal capable of high baud rate o per-at ion arid standard 80x24 display?".

c) The modem in the TAPR kits uses the superannuated, relatively low-performance 2206/2211 chips; t heref ore, it must be ca 1 i brat ed f requent ly and is susceptible to temperature irtduced drift.

d) The software for the TNC is hidden in proms (firmware), undocument ed, and t heref ore not easi ly modi f ied or extended.

e) The rich CP/M eriv i ronment is not directly available for use in file transfers, split screen displays, arid ex t ended command process i ng and logyi ng.

f) I wanted t o leave open the opt i on to adapt the concepts developed for the Z80 urider CP/M to other processors and operating systems for advanced TNCs and network controllers.

g) The TNC firmware is unembellished. There are no d i agnost ic modes, no help modes, nor i s menu dri vert operat ion avai lable.

A Status Report on the Packet Master System

I have developed and have been running a experimental version of the PacketMaster system for the Xerox 820 and the Kaypro line of computers (hereafter to be known as the PacketMaster-820 and the Packet Master-Kaypro, respectively). The program is of a few thousand lines of modularized Z80 assembly code executing under CP/M. The additional hardware is consists of an external Am7910-based modem and NRZ/NRZI conversicm circuitry. Physically, it is simply a board that connects to one of the serial ports.

The system as it now exists is also compatible with the Ferguson Big Board computer line since the Xerox 820, Kaypro, and Big Board are all closely related. It is probably possible to adapt the system to any CP/M computer with a Z80 and a Z80-SIO connected to allow the use of mode 2 interrupts.

AVAILABILITY of the Packet Master System

If you are interested in increasing the Packet I.Q. of your computer by running the Packet Master system on your CP/M cir DOS computer send a S.A.S.E. business sized envelope to me for current informaticm. Give me at least a cursory descripticm of your computer system so I can determine the applicable information to send you! The externa1 modem board is being layed-out presently (2/86) and should be available soon. Us I describe in the next (final) section of this paper-, I am also working, along with some colleagues, on an IBM-PC packet; interface.

Although the CP/M enviroment was useful and adequate to develop and apply such a packet system, better choices are now available. Namely, the massively popular IBM-PC and its myriad clones and compatibles that run one of the many flavors of DOS is probably the most intelligent choice today. In fact, CP/M is a progenitor of DOS; the other strong influence on DOS is Bell Lab's well-loved UNIX operating system. All the development tools of CP/M (assemblers, librarians, linkers, debuggers) are available under DOS. Their function is generally extended because one of the major limitations of CP/M and the Z80--the 64K addressing limitation of the system is eliminated. More functionality, speed, and intelligence may be built into these programs because of the simple fact that there is up to 10x more space available for them to run on a fully endowed PC! This makes high level language programming a real possibility whenever speed of execution is not a major concern. It is always possible to link high level language routines to assembly language routines that perform time-critical functions. The skillful usage of a structured high level language (example: the C programming language) with reasonable care in commenting results in code that is much easier to comprehend and modify/extend.

The Future: The Packet Master-PC

Therefore, I intend to adapt the PacketMaster concept to DOS machines (the IBM-PC, PC/XT, and its legion of clones and compatibles) after the modem and software have been proven in field test rig using Xerox 820s. The Packet Master-PC will consist of a cheap and simple plug-in board for the PC with an auto-configurable BIOS interface. Of course, the software necessary for a high quality, user friendly interface will be included in the system extending the packet services that are now available under CP/M to DOS. Direct BIOS calls will be available to modify the state of the packet I/O system and to transmit and receive packets for the advanced user.

CONCLUSION

I have briefly related some of my thoughts on second generation micros and the advancements embodied in them. The PacketMaster- and PacketMaster-Kaypro are a reality. And by a somewhat obscure and circuitous route, a flexible packet environment for the IBM-PC will soon be a reality, too!

APPENDIX

description of current
PacketMaster system hardware

7910:
The Am7910 is an LSI modem chip capable of both 300 and 1200 baud operation selectable via 4 input lines to the chip. The dual baud rate option is highly advantageous when shifting between hf operation (where 300 baud is the maximum baud rate allowed) and vhf (where 1200 baud is the norm). Its other major advantage is that is crystal controlled and thus needs no initial calibration or adjustments during normal operation. It is not new to amateur radio-- it is used in the Kantronics Packet Communicator. It's original high price (now dropping) and a fair amount of quirkiness in operation, and just plain inertia may have discouraged the TAPR group from designing it into their products. Once these initial hurdles are overcome the Am7910 proves to be an excellent, high performance choice as the basis for a modem without the requirement of any external active filtering.

description of current
Packet Master system software

pack:
main module-- controls the startup of the Packet Control Program (PCP); contains the main program loop; allows a graceful exit of program back to operating system's CCP (CP/M's console command processor)

paklib:
library module for PCP-- contains six routines necessary for the operation of the PCP

the library routines in paklib are--
dispack:
packet disassembly routine
nrmadd:
receive buffer address normalization routine
asspack:
packet assembly routine
cmdprc:
command processing routine
statex:
routine that uses a state table representation of AX.25 level 2 to determine (re)actions in response to external stimuli. Stimuli may be received packets, commands from the *user*, or timeouts of T1, T2, or T3 as defined in RX.25 protocol documents.
basspack:
auxiliary packet assembly routine

SAREX2 SOFTWARE FUR THE TUCSON AMATEUR PACKET RADIO TERMINAL NUDE CONTROLLER TNC 2

Howard Goldstein, N2WX
681 Cardinal Street SE
Palm Bay, Florida 32907-4116

Abstract

The custom modifications of the Tucson Amateur Packet Radio TNC 2 software for the Shuttle Amateur Radio Experiment 2 (SAREX2), and the associated operating modes (robot, meta beacon, logging functions) are discussed.

Background

When Tom Clark W3IWI, president of the Amateur Satellite Corporation, told me of the possiblity of a "Ham in Space" experiment involving packet radio and then held out the opportunity to make use of the TNC 2, I jumped at the opportunity to get involved. We spoke an the phone a few times and arrived at some basic specifications for minimal functionality. Towards the end of the next month (November 1985) I finally had a version suitable for release and simultaneously placed it on the air in Florida and forwarded a copy to the president of the Tucson Amateur Packet Radio Corporation (TAPR) Lyle Johnson WA7GXD, the man responsible fur getting flight-ready hardware together,

Special Operating Modes

ROBOT Mode

Since it was most unlikely that the astronaut ham would be able to devote his or her entire time ta working amateurs, one specification called for an unattended QSO machine, comparable perhaps with the ROBOT mode that was MI some of the Soviet RS satellites. Such a feature would maximize the potential number of amateurs who could make a confirmable, two way contacts with the vehicle,

The package permits up to nine automated contacts to take place simultaneously using AX.25 link layer (version 2.0 or earlier versions). Upon hearing a request to connect from a ground station, the ROBOT assigns a QSO number, and builds a packet which contains the hexadecimal serial number concatenated with a brief, astronaut-settable message. The ROBOT acknowledges the connect request and proceeds to send this packet ten times, or until it correctly receives an acknowledgment frame from the station connecting.

The pint at which the acknowledgement for the serial number and message are received is the point at which the contact is considered a valid two way and logged appropriately. Then the ROBOT will enter the disconnect-attempt state with the calling station, but success or failure on getting the disconnect acknowledgement is not significant to the two way logging function,

Logging

Part of the specification also made it clear that the local terminal (i.e. the one on the shuttle) would not be available for logging the contacts and "heard" data, In this case how on earth (pun intended) can the ground crew ever reconcile claimed contacts with what really happened? How could the lagging data be recovered? At this point it was decided to have the TNC transmit two special kind5 of frames every three minute5 that the ground stations could collect and forward to a central paint for the reconciliation.

One kind of lugging frame is of the format "WA4SIR>WORKED". The information field of this frame contains the last seventeen unique callsigns worked and their associated serial numbers,

The other logging frame, "WA4SIR>HEARD" is similar to the ">WORKED" frame except there is a serial number associated with each distinct transmission of the ">HEARD" frame, and of course there are no contact serial number5 appended to callsigns since only the fact that the station was heard from orbit is significant.

The log types are similar in the respect that a callsign worked or heard that is already logged will not cause a re-ordering of log, This "no update unless needed" philosophy should ease the data reduction chores of those who will be processing the hundreds or thousands of log frames the flight TNC will generate.

Meta Beacons

As the name implies, "Meta" beacon mode provide5 a way for the astronaut to downlink relatively large amounts - 1,792 characters - of information at regular intervals for the packet community at large. Once set up "Meta" beacon mode will continue to retransmit the data indefinitely.

This customization was the simplest, requiring only that a dummy link with the callsign WORLD be recognized internally as one that will always transmit packetized data yet ignore any retry counters (or received frames from WURLD for that matter), Meeting specifications should always be this easy!

Conclusion

Despite popular belief, it IS possible to balance the interests of the programmer (who wants to minimize complexity) with the interest5 of the user (i.e. maximize performance). A thorough specification of objectives goes a long way towards insuring the software delivered does what it was assumed it would be capable of, And a specification developed jointly between programmer and requestor is one usually capable of being met by the desired delivery date,

FEATURES OF THE VADCG TNC+

Douglas Lockhart, VE7APU
Vancouver Amateur Digital Communications Group
9531 Odlin Road
Richmond, B.C. V6X 1E1
(604) 278-5601

Abstract

This paper describes the features and design philosophy of the new VADCG TNC+ (TNC plus) terminal node controller which is the second TNC produced by the Vancouver Amateur Digital Communications Group (VADCG). The TNC+ has many unique features not found on the current TNCs being marketed. These features facilitate Amateur packet radio software development and dissemination and permit the interested user to learn about the detailed operation of the TNC and various protocols. The VADCG TNC+ is an 'open' system as opposed to a 'black box' system.

Background

Members of the VADCG designed the first Amateur packet radio board in 1979 and after some discussion one evening in the living room of my house over possible names to give the device, we decided to call it a 'terminal node controller' (TNC). Although there were a lot of other proposals that evening, this terminology is now used world-wide to describe this commonplace piece of equipment. The same term is now also used to describe equipment with both a controller and modem although at the time the term was coined, it was considered that the modem and controller functions were separate. The original board used an external modem.

When the first VADCG TNC was designed in 1979, 2716s cost $80 each and they were hard to find even at that price. So the first TNC was designed with the less expensive 2708 EPROMs. Time has passed and memory prices and technology have substantially improved. Although the original TNC's lifetime has been extended through a slight modification allowing the use of 2732 EPROMs or by the use of add-on or 'daughter' boards, the time has finally come to replace the board with one with more memory and some new features such as long-term battery backed up RAM which were not possible when the original board was designed.

The original TNC has now been around for over six years and the original purchasers have certainly seen a long lifetime for a pioneer project. Software development is still being done on the original board and software developers using it usually have made available all the source code freely and free of charge for the various programs developed for the board. The tradition of supplying software source free of charge for Amateur packet radio use started with the original software for the first TNC which was written by myself in the spirit of cooperative development. This tradition of free sharing of information is dying out now that Amateur packet radio has been turned over to commercial interests.

Protocols such as V-l, V-2, AX.25 and V-3 as well as various types of repeater and digipeater programs, monitors and debuggers and user interfaces have been provided free of charge to users of the original VADCG TNC. These programs were usually developed on CP/M systems. There is even a system which allows three of these protocols to be resident in the TNC at the same time.

When the original TNC was designed, it was intended to be used as a common development system so that software developers could exchange their programs and sharetheirwork for the commongoalof Amateur packet radio development. It was designed so that developers could use the most common development system available to thematthetime- theCP/M system. It was intended that it would meet the needs of Amateur packet radio developers long into the future. It was designed to operate at speeds up to 64,000 Baud which at the time were thought to be necessary for a useful Amateur packet radio network. It is somewhat of a surprise and a disappointment to the original developers of the TNC to see that almost seven years later, in 1986 that almost all packet radio operation is done at speeds no higher than 1200 Baud. The high speed capability of this board has never been used.

In the intervening years since the VADCG pioneer TNC development, Amateur TNCs have degenerated into mass-produced 'black boxes with canned programs with unavailable source code and little or no information on the internal workings of the software. Some have been stripped of the common development tools such as the ability to display and alter memory. This has presumably been done to keep the inner workings of the software a secret. It is the author's opinion that these closed

systems and an almost paranoid tendency to standardize at any cost suppress the further development of Amateur packet radio and in fact are significant enough that they make it doubtful that the full potential of Amateur packet radio will ever be realized.

In spite of these somewhat gloomy expectations, the VADCG has produced its second TNC - the TNC+. This work has been done in order to encourage more technical development of Amateur packet radio hardware and software by providing a TNC that is easy to develop and exchange programs for and has the capability of high speed operation. But note that the TNC+ is not just intended for developers. It is intended to be used by both developers and end users alike. It is hoped that the end users will realize the importance of using equipment and software that can be upgraded by local developers and not simply opt for the cheapest TNC available at the time.

SYSTEM DESIGN

Although many people helped to bring the TNC+ into existence, the design of the system was done mainly by two individuals - John Spraggs, VE7ADE and myself. It is important to understand the design philosophy behind the VADCG TNC+ as it is not the same as most of the TNCs on the market. Factors which went into the design of this TNC are:

We wanted to modernize the TNC while still keeping faith with the purchasers of the original VADCG TNC designed in 1979. We did not want to negate the original purchaser's investment in hardware or the time spent in getting the system operational,

We also wanted to avoid negating all the effort by many people developing software for the original board.

We wanted to provide a more 'open' system than most of the other TNC manufacturers. The source for most of the software running on the board is available.

We wanted more and better ways to upgrade and distribute new software for the TNC than by replacement of EPROMs. The large battery backed up RAM space combined with selectable write protected areas allows programs to be distributed in five different ways with appropriate software:

1. Programs may be loaded from a 'HEX' file from a computer.
2. Programs may be loaded from a remote computer using a packet radio link,
3. Programs may be transferred directly from one TNC to another using a packet radio link without the aid of a computer.
4. Programs may be loaded over a telephone connection with simple interface hardware.
5. Programs can still be provided in burned in EPROMs as has been done in the past.

We wanted to provide a system that software developers could easily use. With this TNC, the software developer does not have to own an EPROM programmer and eraser. New programs can be loaded quickly from a serial port on the development system. In addition, an operating system using a version of FORTH called STOIC and an assembler is already operating and available which runs on the TNC+ itself. With these tools, a knowledgable developer can program directly on the TNC in either assembly language or higher level Forth type words. This in fact, allows the TNC to operate as an independent microcomputer development system.

We wanted to increase the flexibility and function of the TNC rather than strip it to bare essentials as is being done by others. The parallel port allows use of a printer and terminal simultaneously or provides signals for remote control applications. The serial port supports automatic speed and format selection (Autobaud) over a wide range. The Baud rates are not just the binary rates between 300 and 9600 but go up to 38,400 Baud and down to DC including all standard speeds in this range as well as many non-standard speeds. For example, speeds of 45.45, 110, 400 and 134.5 are supported by the hardware.

Instead of an ion-board modem providing only low speeds of 300 or 1200 Baud, the TNC+ has an industry standard connector to an off-board modem and supports synchronous modems at Baud rates up to 64,000 Baud and asynchronous moderns up to 9600 Baud and both NRZ and NRZI encoding is supported at all Baud rates. In the author's opinion,, the future of Amateur packet radio urgently requires a move to higher Baud rates for both the end-user and backbone links. The TNC+ can use almost all existing standard modems such as Bell 202, 212A, 103, etc. as well as many others which have not seen common use in Amateur radio as yet. (An off-board radio modem is available from the VADCG which supports changes from 300 to 1200 Baud with switch selection and requires no tuning up and is powered directly from the TNC+.)

Compare this for example, to the off-board modem connector in many other TNCs. They frequently do not use a standard connector or standard voltage levels (TTL instead of RS-232), and only support the more expensive synchronous modems and require special signals which are not supplied by standard modems.

We wanted to ensure that the TNC+ had sufficient capacity for expansion. The full 64kB memory support should provide adequate memory for many future packet radio software developments.

HARDWARE SPECIFICATIONS

Timing

The 8085A Microprocessor uses a 4.9152 MHz. crystal to provide a master system clock of 2.4576 MHz. A binary dividing chain provides frequencies from 307.2 kHz. down to 4.6875 Hz. for HDLC controller data rates and software clocking. The timing chain may also be used for interrupting the microprocessor at selected regular intervals.

Memory

The full 64k addressing space is supported and the following combinations of RAM/EPROM are supported by jumper selection:

8k EPROM - 56k RAM
16k EPROM - 48k RAM
32k EPROM - 32k RAM

The RAM below 8000 (Hexadecimal) in the address space may be selectively write protected by jumper selection. All RAM is backed up by a Lithium battery expected to last about 5 years if the TNC+ is powered off. If the TNC is powered on, it should last for the shelf life of the battery which is greater than10 years. 8 sockets for 2764 type EPROMs and 6264LP RAM chips are provided. A Voltage comparator automatically disables memory and resets the microprocessor when the supply voltage drops into an out of specification range. This prevents the microprocessor from writing garbage data into random memory locations during power brown-outs.

Serial Port

Uses an 8250 UART with internal Baud rate generator using industry standard 1488 and 1489 RS-232 drivers to a standard female DB-25S connector. The connector may be configured as either a DCE or DTE(or non-standard connections) with a jumper plug provided. The RS-232 signals supported are:

Transmit Data	TD	*
Received Data	RD	*
Request to Send	RTS	
Clear to Send	CTS	*
Data Set Ready	DSR	*
Carrier Detect	CD	*
Data Terminal Ready	DTR	
Ring Indicator	RI	

The serial port can be used in polled mode or interrupt driven mode by the software. Changes in status of the lines marked with an '*' can generate interrupts. The serial Baud rate, data format, stop bits, etc. are under complete software control allowing for automatic selection of data speed and format (Autobaud). Standard Baud rates supported are: 38,400, 19,200, 9600, 4800, 2400, 1200, 600, 400, 300, 150, 134.5, 110, 75, 45.45. Many other non-standard speeds are also supported by the hardware.

Modem port

An Intel 8273 HDLC/SDLC protocol controller chip is used with 1488 and 1489 RS-232 drivers and an industry standard DB-25S (female) connector supporting the following signals:

Transmit Data	TD	*
Received Data	RD	
Request to Send	RTS	*
Clear to Send	CTS	
Data Set Ready	DSR	
Carrier Detect	CD	
Transmit Clock	TC	
Receive Clock	RC	
Data Terminal Ready	DTR	*
Signal Quality	SQ	
Ring Indicator	RI	
Data Speed Select	DSS	*

The output signals are indicated by an '*'. In addition, +12 and -12 voltages are supplied on pins 9 and 10 of the DB-25S connector to supply power to the modem. The on-board switching power supply has excess capacity to power a modem. A separate header connector provides the above signals at TTL levels along with +5, +12 and -12 voltage levels for an internal modem (inside the same cabinet). Both full duplex and half duplex, synchronous and asynchronous modems are supported by means of on-board jumpers. Asynchronous modem speeds at the following Baud rates are supported by means of on board jumpers: 9600, 4800, 2400, 1200, 600, 300, 150, 75. Other asynchronous modem speeds (such as 400 Baud) are possible up to 9600 Baud if clocks are provided on RC and TC by the modem, Synchronous modems are supported at any Baud rate up to 64,000 Baud. (Baud rate is controlled by the modem.) Note that the Baud rate may be limited by the software being used.

The 8273 transmit and receive functions are completely independent allowing for full duplex operation. Separate receive and transmit interrupts are provided allowing for interrupt driven software.

Parallel Port

An 8255A Programmable Parallel Interface provides 24 I/O lines of TTL level signals on a DB-25S connector. These lines may be programmed under software control in several different modes of operation such as input or output, handshaking, bidirectional and interrupt control. They may be programmed to drive a printer with a Centronics parallel interface or provide control lines for specialized equipment. The parallel port may be operated under interrupt control.

Configuration Switches

An 8-position DIP switch is provided which can be read by software.

Indicator **LEDs**

There are 4 on-board LEDs which can be turned on and off by software. A header is provided for mountfng the LEDs off board if desired,

Trap Switch
Circuitry and a connector is provided to allow for an off-board Trap interrupt switch. (Non-maskable interrupt) This is very useful for a software developer to analyse software failures.

Reset Switch
Circuitry and a connector is provided to allow for an off--board Reset switch or pushbutton. A reset function is automatically performed when the TNC+ is powered on.

Power supply
An on board switching power supply and regulation is provided to supply all voltages needed by the board as well as Power for off board circuitry such as modems. Nominal DC power of +10 to +15 Volts is required but typical operation is between +7 Volts and +18 Volts. A connector is provided to operate the TNC directly from regulated +5, +12 and -12 Volts without the need of the switching power supply components. When the board is not supplying off-board power the measured current consumption without CMOS components is:

290 mA at 17 Volts
340 mA at12 Volts
480 mA at 7 Volts

The on board switching power supply supplies regulated power at the following currents for off-board devices.

5 Volts at 2 Amperes
-12 Volts at approximately 120 mA.

The input power Voltage (+10 to +15 Volts) is also available.

PC Board
7.75 X 8.5 inches (197 X 216 mm.) Double--sided G-10 Glass Epoxy with plated through holes. The board is silk-screened and solder masked.

Upgrade
Users with the original VADCG TNC board should note that this board is exactly the same size as the original board with mounting holes and edge connectors in the same position as the original board. The major components from the original board such as the 8273, 8255, 8250, 8085, 1488s, 1489s, 4024, 74LS373, 74LS132s, 74LS00 and 4.9152 MHz. crystal can be removed and installed on the new board and the old power supply can be used to power it instead of using the on-board switching power supply components. This will significantly reduce the cost of upgrading; to the new board. A special parts kit will be available for those who already have the original VADCG TNC and are upgrading to the TNC+.

Documentation
Since the TNC+ is being supplied as a kit, the documentation includes step-by-step assembly instructions, parts lists and circuit diagram as well as hardware configuration instructions. Because many different types of software are available to run on the TNC+, separate documentation packages are provided for each software package or protocol and for the firmware.

FIRMWARE

The TNC+ parts kit will come with a single programmed 2764 EPROM containing code which will allow bootstrap loading of the battery backed up RAM through the serial port or by a radio link to another TNC+. It contains the Autobaud routines and memory file system as well as drivers for the serial, parallel and HDLC ports. The firmware contains buffer and queue management routines and common routines which can be used by programs in RAM and a monitor program which allows for the displaying of registers and status, the displaying and changing of memory, and the display of the directory of memory files and programs. The firmware allows for all but the Trap and Restart interrupts to be vectored to RAM addresses. The firmware provides for the disabling of the AutoBaud function if desired and allows for selection of the program (protocol) to be entered after a hardware reset of the TNC+. These features are useful when the TNC is connected to a host computer.

SOFTWARE

All I/O port addresses remain the same as in the original TNC board so the many programs written for the original board still run in the new board usually with no modifications required. Much time and effort has been made developing these programs and that effort should not have been wasted.

The following programs are expected to be available at the time this paper is published. Most are available now. Note that several of these programs (protocols) may be installed in the TNC+ at the same time. New programs will become available as they are developed.

STOIC - This is a Forth-like operating system for the TNC+. It contains an integrated 8085 assembler as well as additionallow-level communication words for use of the HDLC/SDLC communications interface on the TNC+. This allows the TNC+to be used as a personal computer and allows the knowledgable user to develop communications programs directly on the TNC+. Although this is not a normal TNC function, the structure and addressing capability of the TNC+ made it easy to provide this operating system.

V-l - Also called the original "Vancouver" link level protocol. This was the first protocol in widespread use for packet radio in the U.S. and Canada.

Although superceded by the more generalized V-2, V-3 and AX.25 protocols nowadays, it is nevertheless the most efficient protocol in terms of overhead and may well be the best choice for point to point half duplex links and for the current type of satellite communications.

v-2 - An efficient link level protocol also developed in Vancouver and used in several countries. The specifications for this protocol were published in the proceedings of the third ARRL Amateur Radio Computer Networking Conference in the paper titled, "A New Vancouver Protocol." This implementation supports the International Standard X.3 and X.28 user interface protocols as well.

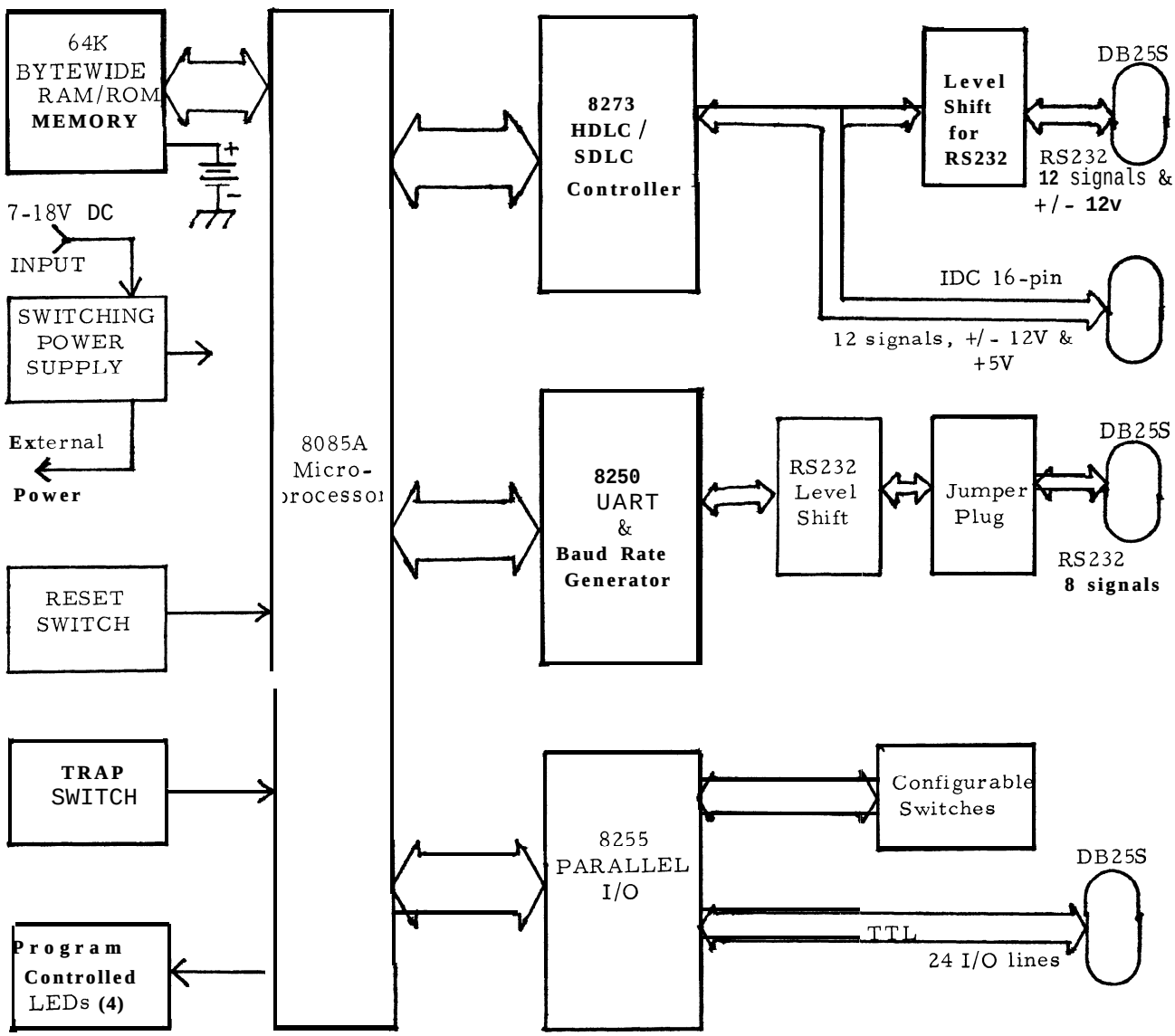
V-3 - This is a experimental new state-exchange link level protocol capable of supporting multiple links simultaneously. Like V-2, it uses the X.3 and X.28 protocols as well.

AX.25 - This is currently the most popular protocol in use. This version supports multiple repeaters.

Note that the source code for most of these programs will be made available on diskette.

SUMMARY

The author is grateful to those volunteers whose efforts have made the TNC+ available. It is hoped that the availablility o.f this board and its software will accelerate the technical development of Arnateur packet radio and give interested Amateurs a tool which can be used to learn the details of packet radio communication.



TNC+ BLOCK DIAGRAM

THE NETWORK USER INTERFACE

David W. Borden, K8MMO
Director, AMRAD
P.O. Drawer 6148
McLean, VA 22106-6148

Abstract

This paper discusses the user interface, the way the human operator interacts with the amateur digital network using their Packet Assembler/Disassembler (PAD) or older Terminal Node Controller (TNC). In this day of increasing features and networking, the packet radio user is expected to remember more and more commands to communicate on the amateur radio packet network. The user desires more and more features and yet wants to keep it simple. The past two years has seen more and more TNCs come on the market, but no standardization of user commands. Now that software is starting to appear to transform these TNCs into Packet Assembler/Dissamblers (PADs), the time is here to make this right, to standardize the user interface using X.3 and X.28 protocols.

Introduction

In the beginning, there was Doug Lockhart. Several Washington area amateur radio operators purchased his TNC and began packeteering (actually framing). As is described in his excellent paper "X.3 AND X.28 PROTOCOLS FOR TERMINAL NODE CONTROLLERS", presented at the Fourth Computer Networking Conference, this TNC software allowed two commands. Control-X caused a connection to be made and Control-Y caused a disconnect. This had a number of drawbacks, the most noteworthy of which was that you had no control over the device for dynamic situations. When you wanted to work HF and should have changed your maximum frame size, or re-try count, or FRACK, you did not even know that such things were possible. You could do three things, monitor the channel seeing everything, communicate in the unconnected mode with no acknowledgements that your packets were received, or communicate in connected mode with one other amateur. Those were the simple days. But, packet radio matured, more control was required.

Doug's Vancouver code was then subjected (the only correct word) to many changes regarding the user interface. Hank Magnuski and his San Francisco gang did most of it until we in AMRAD learned the trick. We hated Guru assigned numbers for addresses (limited to 253) so we implemented the Terry Fox addressing scheme (actually amateur callsigns). Hank. made a digipeater using a STD Bus computer. This required changes to allow the direct frame not to interfere with the repeated frame. Transparent mode was added to allow computers connected to our Vancouvers. Ax.25 got coded, requiring more user changes. The real breakthrough on Vancouver was the AMRAD Daughter board, giving us more PROM space. Code could grow even more without giving up an old feature. More commands were added. Today you have to consult WD5DBC, Howard Cunningham, the keeper of the Vancouver board AX.25 code, to tell you all the commands possible. The basic San Francisco code remains, however. To get to command mode, you type a Control-P and then follow it by some character, like C for connect or D for disconnect. Here you get a hint of the problem, however. As more features were added, more commands were required to control them. In a very short time you can forget how to get into DEBUG mode, or turn MONITOR off, or string digipeater callsigns. Typically what you did then was give your Vancouver away to your club for digipeater use and get a TAPR TNC. Surprise! You have a whole new set of commands to learn.

Menu Driven TNCs/PADs

Since I own a Macintosh and friends of mine own Commodores, I realize that some user interfaces do not require strange commands to deal with the amateur packet radio network. But these computers are really acting as super-smart terminals and provide the desired TNC with the commands required, transparent to the user who points with their mouse or whatever. The intent of this paper is to ignore those menu driven units as the majority of packeteers have TAPR or TAPR clone units and use terminals or computers with less sophisticated user interfaces to communicate. Whatever changes are made to the TNCs/PADs, the menu driven software will be changed to keep up and to keep the whole thing transparent to the user.

Features Requiring Support

Basically the user wants complete control over everything while keeping it simple (just like computer users everywhere, they want control of the world with two commands). I have identified the following actions requiring command support. There are probably others, but these are supported by my TAPR2 PAD running experimental NET1.0, AX.25/AX.75 Level 3 code. Note, I only define new network commands, other older commands are in the TAPR documentation:

a. Getting to the command mode and back

```
Control-C
CONVERS
```

b. Monitoring the channel or not with features

```
MALL ON
MALL OFF
MFILTER n1[,,n2[n3[,n4]]]
MONITOR ON
MONITOR OFF
MHEARD
MHCLEAR
MSTAMP ON
MSTAMP OFF
```

c. Level 2 Connection with another station or packet switch (essentially "framing") with features

```
CMSG ON
CMSG OFF
CONMODE CONVERS
CONMODE TRANS
CONNECT call ±VIA call2[,call3...,call9]]
CONOK ON
CONOK OFF
CONPERM ON
CONPERM OFF
CONSTAMP ON
CONSTAMP OFF
CPACTIME ON
CPACTIME OFF
CTEXT text
```

d. Level 2 Multiple Connection (esentially multiple framing)

```
LCSTREAM ON
LCSTREAM OFF
STREAMSW n
STREAMCA on
STREAMCA off
STREAMDB on
STREAMDB off
USERS n
```

e. Level 2 Disconnection

DISCONNECT

f. Level 3 call placement (essentially "packeting")

CALL [callsign] [@address] - Try to place a Level 3 call

g. Level 3 call servicing

CRESET - Reset Level 3 flow control on my existing call

INT - Cause an interrupt received indication to be displayed at the far end of my call

h. Level 3 call clearing

CLR - Tear down my Level 3 call (like hanging up my phone so I can redial)

i. Enabling/Disabling Digipeating

DIGIPEAT ON
DIGIPEAT OFF

j. Enabling/Disabling Packet Switching (TAPR acts as packet switch)

SWITCH ON - Cause my TAPR device to act as a packet switch

SWITCH OFF - Return my TAPR device to normal user status from packet switch status

SWITCHID nnnnnnn - The ID number of my switch is nnnnnnn

k. Enabling/Disabling Level 3 Network Operations

NETWORK ON - Place my TAPR device in PAD status, vice TNC status

NETWORK OFF - Return my TAPR device from packet PAD status to the old frame TNC status

l. Setting/changing my Level 2 address (callsign)

MYCALL call

m. Setting/changing my Level 3 network address (using X.121NA)

MYNADDR nnnnnnnnnn - My Level 3 network address is nnnnnnnnnn

n. Determining my status as regards connection, or setting of any parameter

CSTATUS
DISPLAY

o. Setting/changing of PAD transmission parameters

DWAIT **n**
FRACK n
PACLEN n
RETRY n
TRIES n
TXDELAY n

p. Enabling/Disabling Level 2 Trace and Level 3 Trace for debugging

NETBUG ON	(Level 3)
NETBUG OFF	(Level 3)
NETTRACE ON	(Level 3 - Trace packets)
NETTRACE OFF	(Level 3)
TRACE ON	(Level 2 - Trace frames)
TRACE OFF	(Level 2)

q. Miscellaneous nice to have features

8BITCONV ON	8BITCONV OFF	AUTOLF ON
AUTOLF OFF	**AWLEN n**	**AX25L2V2** ON
AX25L2V2 OFF	**AXDELAY n**	**AXHANG n**
BEACON EVERY n	BEACON AFTER **n**	BKONDEL ON
BKONDEL OFF	**BTEXT** text	**BUDLIST ON**
BUDLIST OFF	**CALIBRA**	**CALSET**
CANLINE n	**CANPAC n**	**CHECK n**
CLKADJ n	**CMDTIME n**	COMMAND **n**
CR ON	CR OFF	
DAYTIME yymmddhhmm		
DAYUSA ON	DAYUSA OFF	DELETE ON
DELETE OFF	**ECHO ON**	**ECHO OFF**
ESCAPE ON	**ESCAPE OFF**	FLOW ON
FLOW OFF	**FULLDUP ON**	**FULLDUP OFF**
HEADERLN ON	**HEADERLN OFF**	**HID ON**
HID OFF	**ID**	**LCALLS**
LCOK ON	**LCOK OFF**	**LFADD ON**
LFADD OFF	**MAXFRAME n**	**MCOM** ON
MCOM OFF	**MRPT ON**	**MRPT OFF**
MYALIAS	**NEWMODE** ON	**NEWMODE OFF**
NUCR ON	**NUCR OFF**	**NULF ON**
NULF OFF	NULLS **n**	
PACTIME EVERY **n**		
PARITY **n**	PASS **n**	**PASSALL** ON
PASSALL OFF	**REDISPLA n**	RESET
RESPTIME n	SCREENLN **n**	**SENDPAC n**
START n	STOP **n**	TRANS
TRFLOW ON	TRFLOW OFF	TXFLOW ON
TXFLOW OFF		
UNPROTO call1 [via **call2[,call3...,call9]]**		
XFLOW ON	**XFLOW OFF**	**XMITOK ON**
XMITOK OFF	**XOFF n**	**XON n**

These available features are a big step forward from my Vancouver, but still lack standardization. The TAPR way of addressing these features is probably the widest know in the packet community, but is not standard. Other TNCs do not use these TAPR commands unless they use the same code (essentially TAPIR "clones").

Time To Implement

Reference to Lockhart's paper on X.28 and X.3 (he points out it is premature to discuss X.29, a Level 5 issue) reveals details about those two protocols and is excellent reading. The intent of this paper is not to attempt to improve on Doug's paper or even bring it up to date (For example: Interrupt and Reset commands are being used now>. But rather, this paper is an appeal to implement these protocols on TAPR boards and clones. The time for implementation is now, since Level 3 has arrived. Users have to learn new commands to give their PADs anyway, so why not go to the standard now?

Switch Supplied Network Access PAD

There is a simple way out of the need for standardization. The packet switch can be made super-smart, like the Macintosh or Commodore software, and act as a big gateway to the network for all current users. The switch looks at what gibbirish you are typing to it and does the right thing (tries to place your call, etc.). Howard Goldstein has done this for his TAPR packet switch (thats a normal TAPR board with experimental software, allowing it to act as a packet switch - **not the** NNC). A current user having a normal TAPR1 or TAPR2 board and current software (no new ROMS required) can tie into the Level 3 network. The packet switch performs two functions then, acts as a PAD (accepting TAPR Level 2 frames and pushing Level 3 packets out to the network, receiving Level 3 packets back and giving them back to the TAPR users as Level 2 frames). This solution will be required in the near term, but ultimately, users should have PAD software in their devices instead of TNC software.

One Last Appeal for Standarization

I know there will be plenty of room for software in the modern packet switch. But the reason X.28, X.29 and X.3 were written orginally was so that a user could travel anywhere in the world, walk up to a PAD, and use it. I know from experience that it takes a while to learn a new set of commands to do the same old thing you have been doing all along on your previous packet hardware. Why not relearn our packet commands just one more time? As pointed out by Doug in his paper, we can add commands that are just not covered in the protocols but are demanded by amateur radio's strange requirements. But when we want to show link status on all streams, let us use STAT, not CSTATUS. When we want to reset our Level 3 call, the command to do it is RESET, not CRESET (another command can be found for resetting the device, warm and cold). Actually a study of the current TAPR commands reveal that most of the changing required to go to the X. protocols are centered in the parameter setting area.

Conclusion

Now is the acceptable time to become converted to the X. protocols for our amateur packet radio user interface. To deal with the enhanced network, we will all have to learn new commands anyway. Smart packet switches will insulate us from this for a while, but the ultimate answer is to do it in the standard fashion. We of AMRAD will try to do our part in software we provide with our PCPAD board, but that would not have the impact that changing the basic TAPR PAD software would have.

AMATEUR NETWORK ADDRESSING AND ROUTING

Terry Fox, WB4JFI
President, AMRAD
1819 Anderson Rd.
Falls Church, VA 22043

Abstract

Now that we have actual packet switches on the air, it is time to take a hard look at the addressing and routing schemes proposed for use at the Network Layer. In this this paper I give my impression of how I believe addressing and routing will evolve. I will also recommend the use of certain "direction implicit" addresses as an interim step of Network address and routing operation.

Background

There are two major types of devices involved in Network Layer connections within the Network. The source and destination end-points are where the packets enter and leave the Amateur Packet Network, and the transit packet switches (if any) are the intermediate devices that pass the data between the these end-points. In order for each of these devices along a Network Layer connection to understand where packets came from, and where to send them, the various devices along the connection must have a unique descriptor. This is analagous to the source, destination, and repeater addresses in the AX.25 Level 2 Protocol addressing, where the source and destination end-points are the source and destination AX.25 Amateurs, and the packet switches are the digipeaters that allow the two Amateurs to connect to each other.

In the AX.25 Level 2 design, we used the same address technique for the end-point stations and the digipeaters, Amateur callsigns. I suggested the use of Amateur callsigns because they were already assigned to all Amateur stations by the FCC, removing the requirement for some other organization or group of organizations to assign unique addresses.

Since the digipeaters also used Amateur callsigns, their use and identification was made easier. To go through the local digipeater, all an Amateur had to do was specify the digipeater's callsign. If more than one digipeater was required, the additional digipeater callsigns were added at the end of the previous one, forming a list of digipeaters that the Level 2 frame was to pass through. The spec allows for up to eight digipeaters to be used in this fashion. Since the source Amateur has to know the exact list of digipeaters in the proper sequence when the connection is requested, this is called explicit source routing information. Yes, the route comes hand-in-hand as part of the addressing scheme. Even though these are two, separate functions, they are conveyed by the same piece of information, the digipeater callsigns.

Since a similar architecture exists at the Network Layer, we may be able to use some of the same techniques for it as we did at Level 2.

Names vs Addresses

We are all familiar with the concept of names and addresses. Your name is how you are uniquely identified as a person from the rest of the people in the world (well, not quite uniquely, but usually adequately identified). Your address describes where you can be found if some other poerson wishes to communicate with you (in whatever form). It can be argued that the addressing information can imply some additional information, how to get to the named individual (routing information). Sometimes the routing information cannot easily be implied, but must be explicitly given. An example of this is when you give someone directions to your location because they do not have a map.

Such is the case in Network addressing. The Network address may contain the "name" of the Amateur as the callsign of the Amateur that communications is being requested with, the "address" of the Amateur, which would be where in the Amateur Network the named Amateur can be found, and implied by the address information, the route necessary to make the connection to the Amateur. An alternative to the implicit routing would be to use some form explicit routing, similar to the digipeater system mentioned above.

The reason I have the above "philosophical" discussion is to indicate that Network Layer addressing may be called upon to convey more than the simple address information it's name implies. With the FCC no longer maintaining the fairly rigid callsign-to-geographical-area mapping they once did, simply using destination Amateur callsigns may no longer convey enough information to process a connection request. Also, while the Amateur's callsign is a good "handle" for the human to understand, it may not be the best method of identifying someone or something for the network.

Types Of Packet Devices on the Network

The Amateur Packet Network will have several different types of devices running on it. There will be user Amateurs, end-point switches, and transit switches in the broadest terms, with the user Amateurs broken up into smaller sub-sections. Not all of these devices will want or need the same Network addressing information. As the Network develops, less information will need to be present or implied in the addresses. Let's see what information needs to be used in the case of each of the above named devices.

I will start with the user Amateur, since we can all relate to him or her. To start off, let's say a user Amateur (Amateur A) wants to establish a dialog to another user Amateur (Amateur B) directly (without the use of any other device). The only address information needed for this is the "name" of the Amateur B. Simplicity itself. Figure 1 indicates this wonderful scenario.

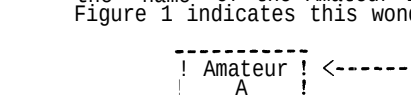


Figure 1. Amateur A to Amateur B Direct

What if Amateur A does not have a direct RF path to Amateur B? Some other device must be used to allow the two to communicate. (Those of you who said a digipeater will stay after this paper to clean the erasers..) Since both Amateurs can communicate with a common end-point packet switch (Switch A), Amateur A requests a connection with Amateur B through Switch A. This is diagrammed in Figure 2.

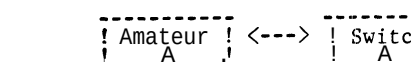


Figure 2. Amateur A to Amateur B-Through Switch A

At the beginning of the Amateur Network development cycle, Amateur A will have to

explicitly tell Switch A that a connection is requested through it to Amateur B. As the local area Network switches develop, Switch A may be able to look in a User directory database, find out that it services both Amateurs, and automatically attempt to make the connection between the two.

If Amateur A and Amateur B are far apart, they may not be able to communicate through a common switch. Instead, the Network connection will have to be made through a series of switches. This is shown in Figure 3, at the end of the paper. At the start of the Amateur Network, Amateur A will now have to know (and pass along) the following:

1. It's own Network end-point Switch name and/or address. This is treated as Amateur A's address by the Network.

2. The route the connection must take to get to Amateur B's end-point switch, and the name and/or the address of all transit switches in that route.

3. The name and/or the address of Amateur B's end-point switch. The Network treats this as the address of Amateur B.

4. The name of Amateur B.

That looks like a lot of information to know, let alone pass along. Actually, items two and three can be though of as the same item, with the first and last name/address being the two end-points. Neverless, with a coast-to-coast connection taking up to 100 switches over VHF, there has got to be a better way. I will call this system Method 1A.

A method of lessening the amount of routing data passed in the packet would be to use addresses for the end-point user Amateur that imply where he/she is located. This way, rather than having to explicitly tell the route, Amateur A could say the equivalent of "Amateur B is in San F'rancisco, CA, USA", Get me there the best way possible. This system will be referred to as method 1B.

The next step in the Amateur Network development will be the capability (through some mysterious magic) of switches to either know automatically, or be able to find the route without getting it from the user Amateurs. One method of doing this would be to have all end-point switches maintain databases containing routes to all other end-point switches. Another method would be to have several Route Server devices spread throughout the Network that the end-point switches would uery whenever they need routing information. The 4ormer has the advantage of speed, while the latter allows the switches to be smaller, and allows for dynamic routing of connections to reduce congestion of frequently used pathes.

Using either system, Amateur A no longer needs to know and specify exactly how the packets get to Amateur B, but rather just what the two end-point names and/or addresses are, and Amateur B's name. Again, the Network treats the two end-point switches as addresses to the Amateur stations (where the Amateurs are on the Network). I call this system method 2.

Method 2 is similar to how some of the present packet networks operate. When Person A wishes to communicate with Person B, Person A dials a local phone number (end-point Switch A address), and asks for a connection to Person B at end-point Switch B (using end-point Switch B's name). As an example in the Amateur Network I might type: connect to WB4JFI-4 @ WB4JFI-5, where WB4JDI-4 is my station at work, and WB4JFI-5 is the end-point switch it normally monitors.

As the Amateur Network progresses even further, all Amateur A may need to do is indicate that he/she wishes to communicate with Amateur B. The Network itself will maintain a list of all Amateurs on the Network, with the end-point switch that normally serves each Amateur as t-hat Amateur's "address", If an Amateur is" away from home", he/she can leave a packet-forwarding address of a different end-point switch on his/hers "home" switch. This is called method 3.

Emphasizing the above Network development stages from the end-point switch side yields the following information requirements.

At initial Network development, the end-point switches will be relatively dumb regarding the "connectivity" of the 'Network. All routing, information will have to come from the user Amateurs in one form or another. Using method LA means that the user Amateur will explicitly tell the end-point exactly which switches to use to make the connection, and in what order. The end-point switch would use this information to build the connection to the destination end-point switch, using any transit switches indicated.

The use of method 113 means the user will inform the switch how to make the connection by adding some "directionality" to the address information supplied. The end-point switch would then make some routing decisions on it's own based on the direction information supplied. It would then pass the packet to the switch it thinks is in the right direction, possibly adding it's name or address to a list somewhere in the packet.

The next step along the Network development cycle (using method 2) would allow the switch to either maintain it's own list of routes to all other end-point switches, or have access to one or more "Route Server' devices (considered User devices by the Network). When an end-point switch receives a connection request from an Amateur, it will look up the indicated destination end-point switch (really, the other Amateur's Network "address") in it's routing database, or inqure of the Route Server the best route to the destination end-point switch. Once the routing information is obtained, the source end-point switch will then attempt to build the connection, going through any transit switches indicated.

After further Network development it may be possible for the end-point switches to find out which end-point switch normally serves as "home"' for the destination Amateur (once again, the destination Amateur's Network "address"). This is method 3, and will will most likely work by having 'User Directory Servers" available on the Network. They would work much like the 'Route Server" except that they would contain the "home"' address of a11 Amateur stations that have appeared on the Network. When a connection request to a destination Amateur is received, the source end-point switch will ask the User Directory Server which end-point switch serves that destination Amateur, and then find the routing information to that end-point switch, and then make the connection, using any transit switches indicated in the routing information.

Transit switches are switches that connections are made through, but not not ended at (except for maintenance connections). As such, the transit switch function does not have any users to worry about. I should mention that end-point switches can function as transit switches also, but the end-point switch functions can be treated as separate functions.

In the beginning of the Amateur Network, transit switches will operate in one of two modes. Using method 1A from above, if a connection request to the transit switch is received from another switch (either another transit switch, or an end-point switch) that contains explicit routing instructions, the transit switch should attempt to pass the connection request along to the next switch indicated. If that connection is not possible, the transit switch should return a packet to the source end-point switch (via the reverse of the path specified) indicating the reason (switch failure, wrong path, etc.).

If a transit switch receives a packet containing implicit routing in the form of directional information, method 1B described above, it should attempt to send the packet in the proper direction, using a "best guess" algorithm. There will most likely be a list somewhere in the packet for the transit switch to append it's identity, so that the destination end-point switch! knows how to reach the source end-point switch.

As the Network progresses, methods 2 and 3 from above will be used more and more. If source-routing is used, this may relax the requirements place on transit switches, since they may no longer need to make routing decisions, but rather just pass connections through them based on routes created by end-point switches.

If the Network does not use source-routing, the transit switches may have to make more routing decisions. This reduces the overhead in the packet that contains routing information. It also allows some degree of Network connection flexibility, since decisions about routes are made along the way, possibly bypassing bad areas of congestion. This will add to to the complexity of the transit switch, since it must now have enough information available to it to make intelligent routing decisions.

Another method of reducing route information overhead is to compress the route information. When the Network gets larger, it may take one hundred or more switches to amke the proper connection. Since there isn't enough room in the packet headers for this amount of routing information (even if we remove the user data). Some of the routing information will have to be "compressed" into a smaller amount of data. This compressed data may have to be expanded by transit switches in order for them to obtain the indicated route information.

Who Gets Assigned What

Another area that should be discussed is who should be assigned Network Layer addressing information, and what kind of addressing information do they need to be assigned. Huh? Let me try to clarify this a little bit.

In the above discussion, I mentioned that the involved user Amateur stations will be indicated by their "names", most likely callsigns. The Network "address" of those Amateurs could be the end-point switches they either are presently on, or frequent. This being the case, those end-point switches now have two means of identification, their own "name" and as an "address" of a user Amateur.

The transit switches also can be identified by their "name" (for maintenance connections), as part of a connection through them their "address", and for routing information their location, once again their "address".

How and what we use to specify each entity may not be thought of as important at first, but if done correctly can help tremendously to ease the growing pains of the Amateur Packet Network.

User Amateur Identification

This one is fairly simple. I feel we shold identify the user Amateur with the FCC assigned callsign and an SSID, similar to AX.25 Level2. In addition, using method 1B above, it may be necessary to add some directional information for the Network to base it's routing decisions on. The possible types of directional information used will be expanded on shortly.

End-Point Switch Identification

When an end-point switch is itself the destination of a connection, the Amateur callsign plus SSID of the switch should be used.

When an end-point switch is used as the source end-point, it should be identified by its callsign plus SSID. Since it is the source of the connection request, all stations will know how to retrace the path to it. This does not preclude the use of alternate paths for the return connection once the network is smart enough to allow such a thing to occur. By then, routing decisions will be made using methods 2 or 3 above anyway.

If an end-point switch is the destination end-point (Network address of the destination Amateur), it's callsign plus SSID should be used, if known. Additional information may be required as follows:

Under method 1A above, since the whole path is explicitly given, no other information needs to be given.

If the callsign of the destination end-point switch is not known (under method 1B above), some directional information must be passed on, so the Network can make routing decisions.

If the callsign of the destination end-point switch is known, it should be sent along with the direction information, reducing the chances of more than one end-point switch answering the connection request.

Using method 2 and 3 above, no additional information needs to be supplied, as the network has the resources to find the routing information from the switch name (the end-user's address).

If the destination end-point switch is part of a path that was compressed as mentioned above, it may need to expand all compressed routes it receives to make sure it is not the destination.

Transit Switch Identification

There are two reasons to need to identify a transit switch. The first is if a maintenance connection is required. This is in effect making the transit switch an end-user rather than a transit switch, and as such the callsign plus SSID of the switch should be used.

The other case is when the transit switch is being used for it's normal, inter-switch communication function. This case is broken down as follows:

If method 1A is being employed, the transit switch should check for it's callsign/SSID in the list of explicit stations. If it's callsign/SSID is found it should attempt to pass the connection to the next switch on the list.

If method 113 is being used, the transit switch look at the end-point directional information supplied, make a best-guess how to route the packet, add it's callsign/SSID to the end of the switch list located in the packet, and send the packet along the route it calculated.

If method 2 or 3 is used, the switch should be identified by it's callsign for source routing, or possibly as part of a "compressed route" that it expands to test. If source routing is not employed, the switch may have to look in it's routing table for the next switch in the route, whose callsign/SSID it uses to pass the connection request along.

With the above background, it is time to look at some of the actual address schemes suggested by the amateur community over the last year.

Address Methods Suggested

It would be difficult for me to list all the possible addressing schemes that have been proposed for use on the Amateur Network. I will concentrate on some of the more popular ones, which are:

1. Callsign only.
2. Callsign plus Area Code/Phone Number.
3. Callsign plus Airport Designators.
4. Callsign plus Zip Code.
5. Callsign Plus Latitude and Longitude.
6. Callsign plus Gridsquares.
7. Assigned numbers based on geographical location.
8. Assigned numbers given by an organization or group of organizations.

Callsign Only

The first one on the list, callsign only, (really callsign plus SSID) is what I hope we all shoot for. It is the most "user friendly"' to the general Amateur community, and therefore the easiest to convince the Amateurs to use. I think we will need some stepping stones to get to that point, however.

Callsign plus Area Code/Phone Number

If additional information must be present, some would say that the easiest system to use is one that is already present. Of the pre-existing systems, the telephone network is probably the best. It provides a large degree of directional information (through country codes and area codes) yet it also offers a high amount of resolution (down to the phone number). Since the numbers are already assigned, and often listed, it should be possible to have fairly quick access to look up needed numbers.

One of the disadvantages is that the phone numbers, and sometimes the exchanges change. This can happen when one moves, often as little as a few blocks. Another disadvantage is that not all Amateurs wish to have their phone number known. For them, it may be necessary to have a set of '"false phone numbers" that could be assigned as needed.. Also, numbering schemes used by other countries may not be as well organized as that of the United States phone system,, particularly those in Europe that have variable length numbers.

The present AX.25 Network code being developed on the TAPR TNC-2 uses area codes and phone numbers, prefixed by another set of digits,, called the DNIC, or Data Network Identification Code. The DNIC code is covered by Gordon Beattie in the 1985 ARRL Computer Networkin Conference proceedings, along with an additional paper found elsewhere in these proceedings. In addition to the area code/phone number information, the TNC-2 code also passes along the source and destination user Amateur Callsigns. The area code/phone number information is contained in the normal AX.25 Level 3 address space, while the callsigns are in the optional facilities section of a call request packet, using the calling and called address extension facilities.

This system seems to be an acceptable scheme to encode both directional information for routing and the Amateur callsign for specific station identification.

Callsign Plus Airport Designators

All major airports around the world now have unique identifiers (that's what the mumbo-jumbo of characters on the tag on your suitcase means). Some have suggested using these as identifiers for a geographical area that an Amateur may be found in. Since airports are scattered randomly about the world, often with large gaps between them, this system would mean that the coverage area of each identifier would vary drastically. The method of assigning airport designators is also not intuitively obvious to the casual observer, making it difficult to extrapolate where one is. In addition, it might be difficult for the average Amateur to find out what the proper airport designator is for unfamiliar areas. I feel this method should not be used.

Callsign plus Zip Code

Once again, the government comes through for us! The postal system has assigned each post office with a unique identifying number, the Zip Code. In addition, if more resolution is needed, we now have Zip Plus Four! Wonderful. Seriously, using Zip codes would work fairly good here in the United States. If one chops off all but the first couple of digits, a reasonable degree of directionality can be obtained. Zip codes are easily found (I believe they are even in the Callbook). The problem with Zip Codes is that outside the U.S. their assignment is not consistant at best, if they are used at all. I think we need a world-wide system for our address scheme, which requires additional "patching" on the Zip Code system.

Callsign Plus Latitude and Longitude

Well, you can't get much better location information about someone than their Latitude and Longitude. Almost. Will we use the East or West Longitude system? Have you looked at your house platte lately? I did a little while back, and found out the numbers there weren't any where near my real Latitude/Longitude. The Latitude and Longitude system would be good except that there is too much information needed to be conveyed if we use them directly, and they can be difficult to find out, especially for an unknown area.

Callsign Plus Gridsquares

In 1980 there was a meeting of European VHF Managers in Maidenhead, Berkshire to establish a world locator system that would accurately define any location in the world with as few symbols as possible. The result of this meeting is the Maidenhead Squares system, often referred to as gridsquares. Gridsquares have been used ever since as the favored system of identifying locations for VHF enthusiasts world-wide. The system is based on placing a grid over the world, with each square twenty deggrees east-to-west by ten degrees north-to-south. Inside each of these squares (called fields) is a smaller grid system, with those squares (called squares) being two degrees east-to-west by one degree north-to-south. If more resolution is needed, the squares are further divided into sub-squares, sized five minutes east-to-west by two-and-one-half minutes north-to-south.

Using gridsquares, a very small section of the world can be described with only six characters. Since gridsquares are based on Latitude and Longitude, directionality can be easily calculated (without having to know Lat/Lon information). Gridsquares can be chopped off at any of the three size boundries to provide the amount of direction information necessary.

The disadvantage of gridsquares is that they may not be easily found. Either a database of them must be maintained, or the Latitude and Longitude information must be converted to the gridsquare needed. Also, no political boundaries are implied by the gridsquares, so callsigns must be referenced for political boundary information.

When I wrote the AX.25 Level 3 Protocol spec (Third ARRL Computer Networking Conference), I was in favor of using gridsquares for the optional locator information. I am still in favor of using them, and will expand on their use shortly.

Assigned Numbers Based On Geographical Location

See Gridsquares above. Seriously, there are many different methods of assigning numbers based on geographical location. I am not sure why someone would want a system other than gridsquares, but it might be possible to come up with a numbering scheme that is better "computer friendly" for the Network, or more compact.

Assigned Numbers by Organization or Groups of Organizations

One method of assigning addresses sometimes suggested is based on a group of hierarchical organizations assigning binary numbers in ever decreasing geographical sizes. In other words, an international group assigns most-significant digits of numbers to countries. An organization inside each country then assigns the next-significant digits. Then smaller, more regional organizations assign ever decreasing numbers, until each Amateur that needs one gets a number.

I am totally opposed to this scheme for several reasons. First of all, it requires the setting up of a wonderful bureauracy, just what we need. Secondly, magic numbers have to be looked up to gain some idea of location (the bureaucrats won't assign numbers in a rational, geographical method since that would make the job too easy). Third, who is going to want to remember binary addresses? Not me. Fourth, where do we go to find out these numbers. Again, not me.

My Proposal For Network Addressing

With the above information in mind, let me now suggest an addressing system that I feel we can grow with.

As mentioned earlier, the user Amateurs should always be identified by their callsigns/SSIDs. The source user Amateur does not need to have provided any locator information for him/her, except for the source end-point switch identification. Therefore, no gridsquares are needed for the source user Amateur.

In may be necessary on occasion to also indicate where a destination user Amateur is located (method 1B above). If this is the case, gridsquares should be added to the destination Amateur's callsign/SSID.

Source end-point switches should also be identified by their callsign/SSID. If there is a reason to add locator information, gridsquares should be added. I'm not sure there is a need for this, since the path is established from the source end-point switch, it should always be known. For now, callsigns should be enough for the source end-point switch.

The destination end-point switch is a different matter. If the callsign/SSID is known, then that should be used as the address of the destination end-point switch. If the callsign/SSID of the destination end-point switch is not known, or of implicit routing is being used in method 1B, the gridsquare of the destination END USER AMATEUR should be included. This should be in the form of gridsquares, using either the two most significant characters if the destination area is not well known or the first four gridsquare characters. Using all six gridsquare characters may provide too much resolution for finding the proper destination end-point switch.

Transit switches should always be identified by their callsign/SSID. It is unclear why one would want to identify transit switches in any other method, other than forgetting one transit switch callsign/SSID when using explicit routing. In any event, callsign/SSID's for transit switches.

How Much Room is Available For Addresses

In AX.25 Level 3, there are three places addressing information may be placed. The first is in the calling and called DTE address fields. Each of these may be up to sixteen nibbles long. I specify nibbles because the DTE addresses are still thought of as numbers, so each nibble is capable of holding one BCD coded digit. This means binary encodings greater than 1001 are not allowed. What a pain.

The second place address information is allowed is in the calling and called address extension facility. Each of these facilities may contain up to 32 nibbles of address information. Once again, the BCD encoding method is imposed. At least here we have room to breath.

The third place where address information can be found is in the two optional routing facilities of AX.25 Level 3. We added these Amateur specific facilities to allow us a mehtod of conveying routing information as the Network was starting. There are two types of routing facilities. The explicit routing facility contains the callsign/SSID of each packet switch the call must go through. The second optional routing facility is the implicit routing facility. The data conveyed in this facility should be some geographical locator information regarding the destination user Amateur.

Now that we know how much room we have, let's look at how to encode our addressing information into AX.25 Layer 3 connection request packets.

Proposed Address Encoding Technique for AX.25 L3

The addressing information I plan to use is gridsquares for location, and callsign/SSID for names. The two are encoded differently, since they have different lengths and are located in different places. I propose that gridsquare information (if needed) be placed as the locator information in the calling and called DTE address fields.

I further propose that the callsign/SSID information be placed in the optional facilities fields (making them less of an option).

Encoding of the Gridsquare Information

Both the proposed address methods (my gridsquare and Gordon Beattie's area code/phone number) include the Data Network Identification Code (DNIC) in them as part of the addressing plan. This inclusion may help us in the future, if we need to tie the Amateur Network to a commercial network. Gordon discusses the encoding of the DNIC in a paper in the 1985 ARRL Amateur Radio Computer Networking Conference, and in a follow-on paper in these proceedings, so I will not describe it in depth here. Keep in mind the nibble and octet (byte) numbers I use below are referenced to the first nibble or octet of the address sub-field in uestion, NOT the absolute count from the start of the packet.

The DNIC fits in five semi-octets, or nibbles (including the prefix). A sixth nibble has been added to the end as a compromise between the phone number and grids quare groups. This sixth nibble is encoded as follows at this time:

4321 <--- Bit pattern in nibble 5.

0000 indicates area code/phone number addressing system.

0001 indicates gridsquare addressing.

The seventh: and succeeding nibbles contain the actual location add ressing information, up to sixteen nibbles. This leaves us with ten nibbles to put the gridsquare information. Unfortunately, the BCD requirement raises it's ugly head, so we have to get cute. In my paper on optional facilities for AX.25 Level 3 (Third ARRL Amateur Radio Computer Networking Conference), as part of ANNEX G I briefly discussed a method of encoding the gridsquare information. Gridsquares are described by six alpha and numeric characters as follows. The first two alpha characters indicate the most significant area (the fields), followed by two numeric digits (for the square), followed by two more alpha characters for the sub-square.

If we take the first two alpha characters and divide each character such that bits 4, 5, and 6 are conveyed in the high order digit, and bits 1, 2, and 3 are in the low order digit, we should be able to put each alpha character in an octet!, while staying within the letter of the protocol. Bits seven and eight are redundant in this case anyway, since upper case alpha is assumed. This uses two octets (four nibbles) for both alpha characters, leaving us with six nibbles. The first alpha character will take up the fourth octet, while the second alpha character is in the the fifth octet. Those who remember the old TV Typewriter I should be familiar with the Half-ASCII this six-bit system uses.

Since the third and fourth gridsquare characters are numeric, they will fit nicely in the two nibbles of octet six, BCD coded anturally.

The fifth and sixth gridsquare characters are not really needed. They describe almost too small of a location, since we may not know exactly where the station in question is. If they are needed, since they are also alpha, we must play the same byte-splitting game with them that we did with the first two gridsquare characters. The fifth and sixth characters will be placed in octets seven and eight of the address field, respectively. This fills up the calling and called DTE address fields in the call request type packets.

Figure 4 shows the encoding of the called address fields for a call request packet to the WB4JFI-5 packet switch at 77 Deg, 4 Minutes, and 47 seconds west by 38 degrees 57 minutes and 7 seconds north (the WDVM-TV tower it resides on>. The gridsquare information for it is FM18LW.

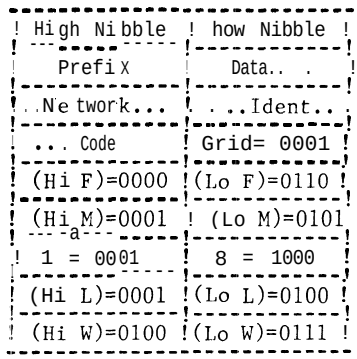


Figure 4. Gridsquare Encoding in DTE Address

Now that we have encoded the gridsquare information, let's put the callsign/SSID information into the optional address facilities.

Callsign/SSID Encoding In Facilities

While I played by the letter of the X.25 Protocol for the calling and called DTE address fields, I think we don't need to go to extremes in the calling and called address extension facilities. Here we can just throw in the Amateur callsign plus a FIVE-bit SSID. The SSID should be justified to the LSB, filling bits 8, 7, and 6 with zeros. The result is that WB4JFI-5 would look like this:

Byte 1	01010111	57 hex = "W"
Byte 2	01000010	42 hex = "B"
Byte 3	00110100	34 hex = "4"
Byte 4	01001010	4A hex = "J"
Byte 5	01000110	46 hex = "F"
Byte 6	01001001	49 hex = "I"
Byte 7	00000101	SSID = 5

If we do run into problems interfacing to commercial networks using this scheme, we can transform these seven bytes into the Half-ASCII system used for the alpha characters of the gridsquares above. There is room to do this, since up to sixteen octets are allowed in each facility.

Implicit Route Facility Encoding

The last encoding scheme to discuss is how we place the locator information in the optional implicit routing facility. I think we should place a marker at the beginning of the facility coded as follows:

00000000 Area code/phone number system.

00000001 gridsquares used.

00000010 Amateur Callsign of destination switch used.

The rest reserved for now.

This will allow expansion for almost any scheme of implicit routing the future may bring, especially for experimentation.

For gridsquare encoding, the six, actual ASCII gridsquare characters of the destination user Amateur can be used, since no BCD rule applies here.

Gridsquare Calculation

There must be some method of finding out the gridsquare information if it is to be used. There are several ways to do this, almost all based on knowing the Latitude and Longitude of the target location. The best way to find out the gridsquare information without knowing the Latitude and Longitude is to ask someone.

Another method of determining the gridsquare of a target location is to use The ARRL World Grid Locator Atlas, available from the ARRL for $4.00. It is a series of maps with the first four characters of the gridsquare information superimposed on them. It also contains a list of the most common cities, states, and countries and their gridsquares. It can be a little hard to extrapolate from these maps the exact grid, so the list is the best bet.

If the Latitude and Longitude is known, the gridsquare information can be obtained, either by computer programs, or by tables. The Latitude and Longitude can usually be obtained to a close enough degree just by looking on a map or atlas (such as the in famous Callbook maps).

There are a couple programs written in Basic and Fortran to do the gridsquare calculations, plus I am working on one written in C. Unfortunately, my program doesn't work so hot south of the equator, so I won't include it here. Instead, I will describe the Gridsquare system and provide some charts for manual conversions. The following information comes from a fine article called "Maidenhead Squares, A World Locator System" by N. A. S. Fitch in the January, 1984 issue of The Shortwave Magazine.

The starting point for the gridsquare system is 180 degrees west Longitude at the south pole. Lettering and numbering runs from west to east, and south to north from that location.

The Gridsquare locator system is made up of six alpha characters. The first, third, and fifth characters are based on longitude, while the second, fourth, and sixth characters are based on Latitude.

The first alpha character is based on twenty degree spacing and identifies the Field in question. This can be looked up in Table 1, which has been converted for west Longitudes.

Degrees West of Greenwich	Field Letter
0 - 20	I
20 - 40	H
40 - 60	G
60 - 80	F
80 -100	E
100 -120	D
120 -140	C
140 -160	B
160 -180	A
180 -200	R
200 -220	Q
220 -240	P
240 -260	0
260 -280	N
280 -300	M
300 -320	L
320 -340	K
340 -360	J

Table 1. First Gridsquare Character

The third character is numeric and is based on two degree spacing from the east side of the defined Field. Table 2 contains the third character information, converted to using the west side for west Longitude calculations.

Degrees West of eastern side of field	Square Number
0 - 2	9
2 - 4	8
4 - 6	7
6 - 8	6
8 - 10	5
10 - 12	4
12 - 14	3
14 - 16	2
16 - 18	1
18 - 20	0

Table 2. Third Digit of Gridsquare.

The fifth character is alpha, and is based on five minute intervals within the previously found square. Table 3 contains the look-up information. Once again, it has been converted for those of us who prefer west Longitudes. Be sure to add 60 to the minutes if your degrees is an odd number.

Minutes West of eastern side of square	Sub-Square Letter	Minutes West of eastern side of square	Sub-Square Letter
0 - 5	X	60 - 65	L
5 - 10	W	65 - 70	K
10 - 15	V	70 - 75	J
15 - 20	U	75 - 80	I
20 - 25	T	80 - 85	H
25 - 30	S	85 - 90	G
30 - 35	R	90 - 95	F
35 - 40	Q	95 - 100	E
40 - 45	P		
45 - 50	N	105 - 110	D
50 - 55		110 - 115	B
55 - 60	M	115 - 120	A

Table 3. Fifth Gridsquare Character Table.

Now, on to the Latitude calculation. As mentioned earlier, the Latitude information is conveyed in the second, fourth, and sixth characeters. The second character is an alpha, and is based on ten degree spacing from the south pole. Table 4 contains the information based on degrees north or south of the equator.

Latitude North of Equator	Field Letter	Latitude South of Equator	Field Letter
+ 80 - 90	R	-0 -10	I
+ 70 - 80	Q	-10 -20	H
+ 60 - 70	P	-20 -30	G
+ 50 - 60	O	-30 -40	F
+ 40 - 50	N	-40 -50	E
+ 30 - 40	M	-50 -60	D
+ 20 - 30	L	-60 -70	C
+ 10 - 20	K	-70 -80	B
+ 0 - 10	J	-80 -90	A

Table 4. Second Gridsquare Character.

The fourth gridsquare is numeric and is based on one degree increments from the bottom of the Field. Table 5 is used to find the fourth digit.

Degrees North	Square Number	Degrees South
+9 -10		
+8 - 9	8	-0 -1 -1 -2
+7 - 8	7	-2 -3
+6 - 7	6	-3 -4
+5 - 6	5	-4 -5
+4 - 5	4	-5 -6
+3 - 4	2	-6 -7
+2 - 3	1	-7 -8
+1 - 2		-8 -9
+o - 1	0	-9 -10

Table 5. Gridsquare Fourth Character Table

The sixth and last gridsquare character is an alpha based on 2.5 minute increments from the bottom of the Square. Table 6 contains the information on it.

Minutes North	Sub-Square Letter	Minutes South
+57.5 - 60	X	-0 - 2.5
+55 - 57.5	W	-2.5 - 5
+52.5 - 55	V	-5 - 7.5
+47.5 - 52.50	U	-7.5 - 15
+45 - 47.5	T	-10 - 12.5
+42.5 - 45	R	-12.5 -15
+37.5 - 42.5	Q	-17.5 - 20
+35 - 37.5	O	-20 - 22.5
+32.5 - 35		-22.5 - 25
+27.5 - 30	L	-27.5 - 30
+22.5 - 25	K	-30 - 32.5
+20 - 22.5	I	-32.5 - 35
+17.5 - 20	H	-35 - 37.5
+12.5 - 15	G	-37.5 - 40
+7.5 - 10	E	-40 - 42.5
+5 - 7.5	C	-42.5 - 45
+2.5 - 5	B	-45 - 47.5
+o - 2.5	A	-47.5 - 50
		-50 - 52.5
		-52.5 - 55
		-55 - 57.5
		-57.5 - 60

Table 6. Sixth Gridsquare Character Table

With the above information and the Latitude and Longitude, anyone should be able to figure out what their gridsquare is, along with those of other Amateur stations.

Conclusion

Not all Amateur stations on the Network will need to supply location information. As the Network builds, less addressing and routing information will be required from the user Amateur, and more will be maintained by the switches. In the interim, I feel we should use the callsign/SSID plus Gridsquare information as needed. I hope the gridsquare tables help to allieviate some of the negative comments I have heard about using them.

References

Fox, T., "AX.25 Amateur Packet-Radio Link-Layer Protocol", ARRL, 1984

Tanenbaum, A. S., "Computer Networks", Prentice Hall, 1981

Beattie, J. G., "Proposal: Recommendation AX.121NA Numbering Plan For The Amateur Radio Network in North America", Fourth ARRL Amateur Radio Computer Networking Conference, ARRL, 1985

Fox, T., "AX.25 Network Sublayer Protocol Recommendation", Third ARRL Amateur Radio Computer Networking Conference, ARRL, 1984

Fox, T., "Packet Formats of AX.25 Level 3 Protocol", Third ARRL Amateur Radio Computer Networking Conference, ARRL, 1984

Fox, T., "Optional Facilities For AX.25 Level 13 Protocol", Third ARRL Amateur Radio Computer Networking Conference, ARRL, 1984

Fox, T., "Annex A Through F For AX.25 Level 3 Protocol", Third ARRL Amateur Radio Computer Networking Conference, ARRL, 1984

Meijer, A. and Peeters,, P., "Computer Network Architectures", Computer Science Press, 1982

Fitch, N. A. S., "Maidenhead Squares A Worldwide Locator System", The Shortwave Magazine, January 1984

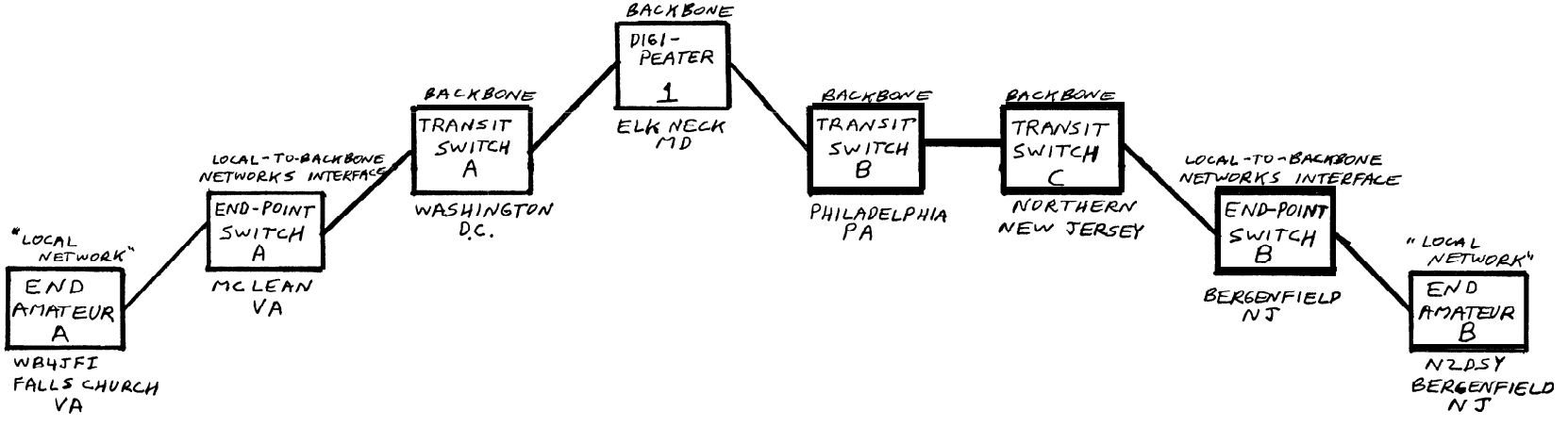


FIGURE 3. MULTIPLE SWITCH OPERATION.

PROPOSAL: RECOMMENDATION AX.121 (1/86)

INTERNATIONAL NUMBERING PLAN FOR THE AMATEUR RADIO NETWORK

J. Gordon Beattie, Jr., N2DSY
Thomas A. Moulton, W2VY

The Radio Amateur Telecommunications Society
206 North Vivyen Street
Bergenfield, NJ 07621
201-387-8896

Introduction

The purpose of this International Numbering Plan is to facilitate the introduction of amateur data networks and provide for internetworking on a worldwide basis.

1.0 Design Considerations

1.1 This proposal does not require, nor preclude governmental involvement in network administration. (i.e. a DNIC for each national amateur network.)

1.2 The International Numbering Plan should permit the identification of a called country as well as a specific network.

1.3 The International Numbering Plan should provide a consistent addressing format when connection is made with or through commercial networks. (i.e. telephone, telex, data networks.)

1.4 A national number assigned to a terminal should be unique within a particular network. This national number should form part of the international number which should also be unique on a worldwide basis.

1.5 Specific national numbers should be easily determined.

1.6 National Numnbers should require minimal administrative overhead to network management and users.

1.7 The International Numbering Plan should provide for substatial spare capacity to accommodate future requirements.

2.0 Numbering System

2.1 The 10 digit numeric character set 0-9 should be used for numbers (or addresses) assigned to terminals in the amateur network. This principle should apply to both national and international numbers.

2.2 Use of the numbering system as outlined in 2.1 will make it possible to interwork with terminals on public telephone, telex and data networks.

3.0 Prefix Codes

3.1 The Prefix Code will signify the type of network indicated by the remaining digits.

3.2 The Prefix Code will be the first digit and should be coded as follows:

0 Amateur Packet Switched Network
1 Public Packet Switched Network
2 \
3 \
4 ___ Reserved
5 /
6 /
7 /
8 Telex Network
9 Telephone

4.0 Data Network Identification Codes

4.1 All Data Network Identification Codes should consist of four digits.

4.2 Each country shall be assigned at least one 3-digit Data Country Code. This in conjunction with a fourth digit will identify up to 10 amateur networks or services within a country. The Data Country Codes are defined in Appendix A/AX.121.

4.3 The CCITT will be the body responsible for the assignment of Data Country Codes. If-required, the CCITT will assign additional DCC's to a country.

4.4 If the national amateur body in a country does not specify the use of the fourth digit in the DNIC, it is to be considered "Reserved" and shall be coded as "0".

4.5 Possible uses of the fourth digit include, but are not limited to, indication of special stations, services or networks.

5.0 National Number

5.1 The National Number shall consist of up to 10 digits.

5.2 The National Number may consist of 11 digits if the national amateur body specifies the use of a Data Country Code instead of Data Network Identification Code.

5.3 Each National Number shall be unique within the country.

6.0 International Number

6.1 The International umber shall consist of the DNIC or DCC and the National Number.

6.2 National Amateur Networks will be capable of interpreting the first four digits of the International Number. This is needed to facilitate routing between networks.

6.3 The use of the DNIC by stations in transit countries would serve as a ready reference for checking third-party traffic handling requirements.

6.4 The International Number may optionally include a prefix code.

7.0 Formats

7.1 Amateur Network Number Format

Prefix + DNIC + National Number

P + DDDD + NNNNNNNNNN = 15

or

Prefix + DCC + National Number

P + DDD + NNNNNNNNNNN = 15

7.2 Amateur Network Number Format (alternate)

DNIC + National Number

DDDD + NNNNNNNNNN

or

DCC + National Number

DDD + NNNNNNNNNNN

Appendix A/AX.121

Data Country Codes

Zone 2

DCC	Country or Area
202	Greece
204	Netherlands
206	Belgium
208	France
212	Monaco
214	Spain
216	Hungarian People's Republic
218	German Democratic Republic
220	Yugoslavia (Socialist Federated Republic of)
222	Italy
226	Romania (Socialist Republic of)
228	Switzerland (Confederation of)
230	Czechoslovak Socialist Republic
232	Austria
234	United Kingdom of Great Britain and Northern Ireland
238	Denmark
240	Sweden
242	Norway
244	Finland
250	Union of Soviet Socialist Republics
260	Poland
262	Federal Republic of Germany
266	Gibraltar
268	Portugal
270	Luxembourg
272	Ireland
274	Iceland
276	Albania
278	Malta (Republic of)
280	Cyprus (Republic of)
284	Bulgaria (People's Republic of)
286	Turkey

Zone 3

DCC	Country or Area
302	Canada
308	St. Pierre and Miquelon
310	United States of America
311	United States of America
312	United States of America
313	United States of America
314	United States of America
315	United States of America
316	United States of America
330	Puerto Rico
332	Virgin Islands (USA)
334	Mexico
338	Jamaica
340	French Antilles
342	Barbados
344	Antigua
346	Cayman Islands
348	British Virgin Islands
350	Bermuda
352	Grenada
354	Montserrat
356	St. Kitts
358	St, Lucia

360 St. Vincent
362 Netherlands Antilles
364 Bahamas (Commonwealth of the)
366 Dominica
368 Cuba
370 Dominican Republic
372 Haiti (Republic of)
374 Trinidad and Tobago
376 Turks and Caicos Islands

Zone 4

DCC	Country or Area
404	India (Republic of)
410	Pakistan (Islamic Republic of)
412	Afghanistan (Democratic Republic of)
413	Sri Lanka (Democratic Socialist Republic of)
414	Burma (Socialist Republic of the Union of)
415	Lebanon
416	Jordan (Hashemite Kingdom of)
417	Syrian Arab Republic
418	Iraq (Republic of)
419	Kuwait (State of)
420	Saudi Arabia (Kingdom of)
421	Yemen (Arab Republic)
422	Oman (Sultanate of)
423	Yemen (People's Democratic Republic of)
424	United Arab Emirates
425	Israel (State of)
426	Bahrain (State of)
427	Qatar (State of)
428	Mongolian People's Republic
429	Nepal
430	United Arab Emirates (Abu Dhabi)
431	United Arab Emirates (Dubai)
432	Iran (Islamic Republic of)
440	Japan
450	Korea (Republic of)
452	Viet Nam (Socialist Republic of)
454	Hong Kong
455	Macao
456	Democratic Kampuchea
457	Lao People's Democratic Republic
460	China (People% Republic of)
470	Bangladesh (People's Republic of)
472	Maldives (Republic of)

Zone 5

DCC	Country or Area
502	Malaysia
505	Australia
510	Indonesia (Republic of)
515	Philippines (Republic of)
520	Thailand
525	Singapore (Republic of)
528	Brunei
530	New Zealand
535	Guam
536	Nauru (Republic of)
537	Papua New Guinea
539	Tonga (Kingdom of)
540	Solomon Islands
541	New Hebrides
542	Fiji
543	Wallis and Futuna Islands
544	American Samoa
545	Gibert and Ellice Islands
546	New Caledonia and Dependencies
547	French Polynesia
548	Cook Islands
459	Western Samoa

Zone 6

DCC	Country or Area
602	Egypt (Arab Republic of)
603	Algeria (Algerian Democratic and Popular Republic)
604	Morocco (Kingdom of)
605	Tunisia
606	Libya (Socialist People's Libyan Arab Jamahiriya)
607	Gambia (Republic of the)
608	Senegal (Republic of the)
609	Mauritania (Islamic Republic of)
610	Mali (Republic of)
611	Guinea (Revolutionary People's Republic of)
612	Ivory Coast (Republic of the)
613	Upper Volta (Republic of)
614	Niger (Republic of the)
615	Togolese Republic
616	Benin (People's Republic of)
617	Mauritius
618	Liberia (Republic of)
619	Sierra Leone
620	Ghana
621	Nigeria (Federal Republic of)
622	Chad (Republic of the)
623	Central African Republic
624	Cameroon (United Republic of)
625	Cape Verde (Republic of)
626	Sao Tome and Principe (Democratic Republic of)
627	Equatorial Guinea (Republic of)
628	Gabon Republic
629	Congo (People's Republic of the)
630	Zaire (Republic of)
631	Angola (People's Republic of)
632	Guinea-Bissau (Republic of)
633	Seychelles
634	Sudan (Democratic Republic of the)
635	Rwanda (Republic of)
636	Eihiopia
637	Somali Democratic Republic
638	Republic of Djibouti
639	Kenya (Republic of)
640	Tanzania (United Republic of)
641	Uganda (Republic of)
642	Burundi (Republic of)
643	Mozambique (People's Republic of)
645	Zambia (Republic of)
646	Madagascar (Democratic Republic of)
647	Reunion (French Department of)
648	Zimbabwe
649	Namibia
650	Malawi
651	Lesotho (Kingdom of)
652	Botswana (Republic of)
653	Swaziland (Kingdom of)

654 Comoros (Federal and Islamic Republic of the)
655 South Africa (Republic of)

Zone 7

DCC	Country or Area
702	Belize
704	Guatemala (Republic of)
706	El Salvador (Republic of)
708	Honduras (Republic of)
710	Nicaragua
712	Costa Rica
714	Panama (Republic of)
716	Peru
722	Argentine Republic
724	Brazil (Federl Republic of)
730	Chile
732	Colombia (Republic of)
734	Venezuela (Republic of)
736	Bolivia (Republic of)
738	Guyana
740	Ecuador
742	Guiana (French Department of)
744	Paraguay (Republic of)
746	Suriname (Republic of)
748	Uruguay (Oriental Republic of)

Appendix B/AX.121

This appendix outlines the options to be used by DTEs operating in the Amateur Data Networks in North America.

1. The Amateur Data Networks of North America will use the numbering format:

   Prefix + DCC + Format + Nat. #

   P + DDD + F + NNNNNNNNNN = 15

2. Each country in the region shall use the Data Country Codes listed below.

   302 Canada
   310 United States of America
   330 Puerto Rico
   332 Virgin Islands (USA)
   334 Mexico

3. The Address Format digit (A) allows amateurs in each network the choice of two formats:

   0 =
   The remaining digits follow the North American Numbering Plan (NANP) used in the public telephone system.

   1 =
   The remaining digits follow the Universal Grid Square System.

3.1 If A = 0 then:

3.1.1 The National Number shall consist of a three digit Area Code, a three digit Exchange Code and a four digit Subscriber Code.

Prefix + DCC + Format + Area + Exchange + Subscriber

P + DDD + F + AAA + EEE + SSSS = 15

3.1.2 The Area Code is mandatory for all networks.

3.1.3 The Area Code must be consistent with the North American Numbering Plan except as directed by the National Amateur Body.

3.1.4 The use of an Exchange Code is strongly suggested. This code may follow the local telephone exchange code.

3.1.5 The use of the Exchange Codes 000 through 199 may be used if assigned by the Network Coordinating Agent for the area.

3.1.6 Service Codes 011, 111, 211, 311, 411, 511, 611, 711, 811, 911 are reserved pending definition by the local Network Coordinating Agent.

3.1.7 Service Codes shall be placed in the Exchange Code field and should not be followed by any additional digits.

3.1.8 The exchange code 555 is reserved for internal network administration and assignment authority is limited to the local Network Coordinating Agent.

3.1.9 The exchange and subscriber code 555-1212 is reserved for regional directory service, Assignment authority is limited to the local Network Coordinating Agent.

3.2 If A = 1 then:

3.2.1 The National Number shall consist of four digits containing the ASCII representation of the first two alphabetic characters, two BCD digits representing the next two numeric digits, and four additional BCD digits.

Prefix + DCC + Format + Alpha + Square + Local Network

P + DDD + F + AAAA + SS + NNNN = 15

3.2.2 Use of the Alpha and Square fields is mandatory.

3.2.3 The each character in the Alpha field will be coded using the ASCII character codes for the capital letters A-Z. This scheme will use two positions per alpha character.

3.2.4 The each character in the Square field will be coded in binary.

3.2.5 The local Network Coordinating Agent must define and assign the four local Network digits.

A PACKET ASSEMBLER/DISASSEMBLER FOR AMATEUR PACKET RADIO NETWORKING

Howard Goldstein, N2WX
681 Cardinal Street SE
Palm Bay, Florida 32907-4116

Abstract

This paper describes the operation af the pratatype Packet Assembler/Disassembler (PAD) functian within the Tucson Amateur Packet Radio TNC 2 the author installed at a tall tawer site near Melbourne, Florida. PADs are usually considered ta be devices. which interface "dumb" asynchronous terminals to packet switched networks. The prototype TNC 2 PAD performs this functian far remote users an the AX.75 netwark af which it is part, while at the same time enabling a new method af establishing nan-level three cannectians which affers Improved performance aver "digipeated" cannectians using the same path.

Intraductian

The capacity af the common 1200 Baud VHF circuit is outpaced in many areas by rapid growth in the user base. As the price of assembled and tested Terminal Node Controllers (TNCs) drap below the $150US mark, this situation can only get worse.

The amateur community taak a pasitive step tawards addressing this prablem through the adoption of versian 2.0 of the AX.25 link layer pratacal. But in arder ta achieve any real net improvement far a system comprising multiple users af a single frequency, thaughts af changing again the link layer may nat hold forth a marginal benefit capable of justifying the tremendous cast that such a change wauld embody.

This author feels that many cangestian problems are successfully addressed thraugh the elimination of what we've called "digipeated" links, and their replacement by a netwark layer - notably, the AX.25/X.75 network layer (1,2,3). Accepting this, one's next question might be "How daes the user base access this reformulated network?"

It is expected that same users will have an AX.25 network layer interface withrn their TNCs. They wan't need an additional PAD facility - their level 3 capability already implies that ane exists, so it is far the benefit of those whose TNC does not include a network layer interface (they'll be referred to as "L2 users" in the balance of this paper) that the PAD was developed.

PAD modes

This prototype PAD has a "NETWORK" made and an "INTERMEDIARY" mode. Which made is used is determined independently far each link, and reevaluated whenever the link is reset.

Network made

NETWORK made provides the gateway between an amateur X.75 "trunking" network, or lacal AX.25 level 3 users, and L2 users. All of these protocols are similar and this simplifies the transliteration between the different pratacals and the services each offers.

Intermediary made

INTERMEDIARY made is virtually a "dummy network" far local L2 users. Where twa L2 users are unable to establish a direct connectian with each other, they may choose to use the PAD ta set up a smart "call", rather than using the dumb digipeat made. Such a choice wauld convey the advantages of lacal hap by hap acknowledgments while allowing same mechanism ta allocate physical link capacity fairly.

PAD aperatian

The PAD presents an interface which is. essentially transparent ta the made in use. There are faur major states (see figure 1) associated with the PAD<->L2 user interface the user needs ta be concerned with.

The selectian state a? is entered whenever an L2 user links with the PAD. The PAD remains in this state until the L2 user specifies a destinatian callsign, an aptianal endpoint switch address, and up ta three optional endpoint digipeaters. If the endpoint switch address is omitted, and the destinatian station is nat linked with the PAD's switch, the INTERMEDIARY made is invoked and a level 2 connect attempt is initiated. Otherwise NETWORK made is assumed and a level 3 channel is selected, and if a free channel is available a call request packet is generated.

State a3 is the basic data transfer state, and is entered upon establishment of an end to end "connectian". The connectian could result from a netwark: call request packet, an L2 user's request to talk arnginating an the same PAD but a different link, or the acceptance of this user's request to talk with another station (i.e. the a? to a3 transition).

State a4 insures that ane or bath L2 user endpoints receive infarmatian about the cause of a PAD mediated connectian "failure" just prior to tearing down the users' links. A transition to the idle al state occurs when this information is acknowledged by the user and the link layer disconnect attempt state is entered,

Other cansideratians

Far operations an a single frequency, explicit (i.e. not directly windaw related) flaw control af a sending L2 user is invoked either upon exceeding an absolute buffer allocation or predictively, at the time when the first byte of a new information field wauld exceed the allocation.

There is same overhead associated with the predictive flaw, but the author believes it is much less than the (implied) overhead resulting fram callisians (when the pad is equipped with only ane part, or INTERMEDIARY made is in use) between acknowledgements from a remote, and the transmission of new data that can not even be buffered until the such acknowledgement is received by the PAD.

Network control, additional physical parts, and other embellishments will be added as time and hardware allow.

Acknowledgements

Deepest thank go to William Hartman N4DNW, 3. Gordon Beattie N2DSY, and Thomas Moulton W2VY, without whose help this project would never have came to fruitian.

References

(1) T. Fax WB4JFI, AX.25 Network Sublayer Protocol Recommendation, "Third ARRL Amateur Radio Computer Networking Conference" proceedings (Newington: American Radia Relay League)

(2) J. G. Beattie N2DSY, T. A. Moulton W2VY, Proposal: Recammendatian AX.121 NA, "Fourth ARRL Amateur Radio Computer Networking Conference" proceedings (Newington: American Radio Relay League)

(3) X.75 is an Internet Protocol (IP), comparable in operation and packet farm at to X.25 level 3

PAD MESSAGES

gator 2 pad 03100305724 part B
enter: call [,digi i [,digi2 [,digi3 3 3 3

to?

-*- signon -*-

*** pad: connection reset

-*- successful connection or reset -*-

*** pad: call cleared, dte originated
*** pad: call cleared, dte busy
*** pad: call cleared, retry limit exceeded far either call or data
*** pad: call cleared, either the station you requested is also using this pad or an unrecognizable TO? entry was received

-*- failure messages -*-

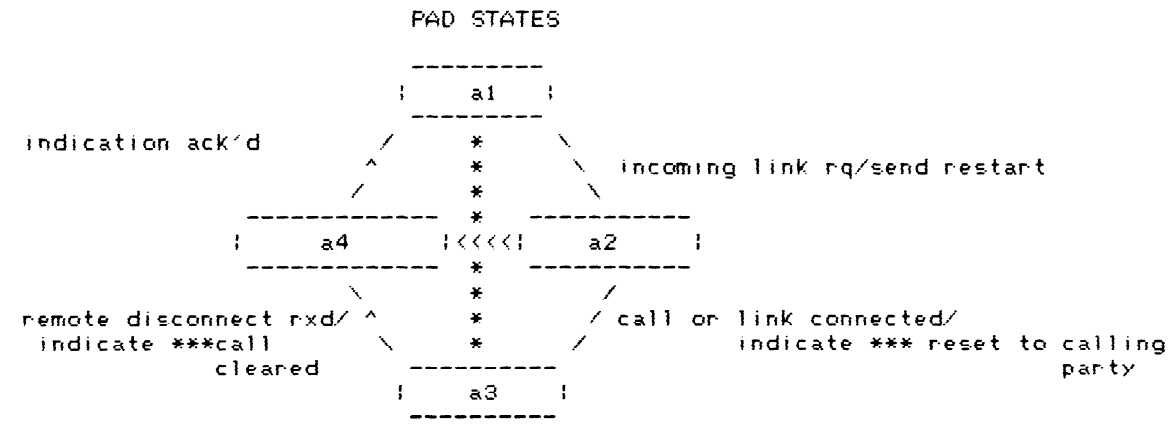


FIGURE 1

A Networking Nods Controller for Amateur Packet Radio

by Lyle Johnson, WA7GXD
c/o Tucson Amateur Packet Radio
PO Box 22888
Tucson AZ 85734-2888

Introduction

Perceiving the need for a standardized hardware environment for developing, testing and implementing Amateur packet radio network and transport level protocols, the author, under the auspices of Tucson Amateur Packet Radio (TAPR), conducted a three month discussion with leading packeteers in the United States during the summer of 1985. The intent was to define the hardware desired in a controller that would be suited to this task. Having reached a general consensus, a system was designed which incorporated most of the suggestions of the group.

The resulting system, the TAPR Network Node Controller (NNC), is currently in its third hardware revision. It is expected to be placed in the field during the second quarter of 1986 for software development and system testing. It is hoped to be available to the general Amateur packet radio community during third quarter 1986.

This paper describes the features of the NNC that make it a logical choice upon which to build an Amateur packet switching system.

Networking Background

The most prevalent protocol in Amateur packet use today, AX.25 Level 2 Version 2, is a link-layer protocol providing single--point to single--point connectivity. In an effort to expand a typical VHF station's RF domain, an expanded address field which allows for digital repeaters (digipeaters) is included in the protocol definition.

Digipeaters are a mixed bag. They have allowed stations to communicate beyond their normal radio horizon, as intended. This feature has been a positive contributor to packet growth. In addition, so-called wide area digipeaters have aided in placing packet-dedicated RF facilities in useful locations for future networking. At the same time, digipeaters have contributed to present-day channel congestion due to the end-to-end acknowledgment used as opposed to using point-to-point acknowledgments along the digipeating path.

Network-level protocols (used loosely in this paper to include appropriate features of transport-level protocols as well), on the other hand, generally provide for hop-by-hop acknowledgment. This helps reduce channel loading, improving channel efficiency and throughput. There are many other features that networking will eventually provide that are beyond the scope of this paper.

NNC Hardware Requirements

A networking controller should be capable of handling multiple simultaneous RF channels to allow it to switch and otherwise (re)route traffic. In addition, it may need relatively large amounts of memory storage to handle buffering of multiple packets. It should also be rugged in order to reliably serve in harsh, remote sites (mountaintops, for example).

An Amateur NNC, in addition to the above requirements, should also be very inexpensive, expandable and use a processor for which there are many familiar and available software tools.

NNC Digital Hardware Description

The TAPR NNC design meets these objectives.

A total of four HDLC ports are included on the digital board. With appropriate modems and radios, this allows up to four radio channels to be used simultaneously. The ports are implemented in hardware for speed, are capable of using full-duplex packet channels and are configured for vectored interrupt operation. In addition, two of the four ports may be operated under direct memory access (DMA) control for very fast channels. These ports are based on the Zilog (tm) SIO/2 chip, a dual-channel device proven in Amateur packet use (eg, the TAPR TNC 2 uses an SIO chip for HDLC operation>.

The microprocessor (uP) chosen for the NNC is the Hitachi HD64180. This CMOS engine incorporates a number of functions usually associated with external peripheral ICs. In addition to the uP core, which is a superset of the familiar Z-80 (tm), a pair of asynchronous serial input/output (I/O) channels, a dual-channel DMA controller, a priority interrupt controller, a programmable 16-bit timer counter, a clock oscillator and a primitive memory management unit (MMU) are all included on this single-chip device.

The MMU allows a total physical address space of 512 kbytes to be utilized by the UP. A paging scheme is employed to divide the physical address space into three logical address spaces. The net result is a fast and relatively efficient expansion of the normal 64 kbyte address space limitation of the Z80.

Memory on the NNC is physically organized into two banks of eight bytewide sockets each,

The lower bank, mapped into the first 256 kbytes of physical address space, includes a hardware wait-state generator that is activated on op code fetches only, allowing the use of inexpensive 250 nSec EPROMs to contain program code. Each of the eight sockets is individually configurable, by means of on-board Jumpers, to utilize a 32 kbyte EPROM or an 8 or 32 kbyte static RAM IC. Each socket consumes 32 kbytes of physical address space, regardless of the XC installed at a particular location.

The second bank of sockets is intended for low-power CMOS static RAM only. This bank is battery-backed, allowing up to 1/4 megabyte of non-volatile RAM to be included on the NNC. A single Jumper on the board configures these eight sockets for 8 kbyte or 32 kbyte RAMs. These parts cannot be intermixed in this physical bank of memory. No wait-states are inserted during op-code fetches in this area of memory, since 150 nSec (and faster) RAM is readily available at low cost.

The IC used to control the battery-backed RAM (bbRAM) circuitry allows testing the condition of the battery every time the NNC is powered up from a battery-backed state. Software routines may utilize this feature to report impending battery failure.

For future expandability, an industry-standard Small Computer Systems Interface (SCSI) port is included on the NNC. The SCSI port may be DMA controlled, or simply interrupt driven. This port should be capable of transferring data at rates up to 1.5 megabytes per second, allowing the NNC to act as a "front end" processor for later high-capacity data concentrators as the Amateur packet network evolves-

A pair of eight-bit parallel ports are included in the NNC. One is configured as a Centronics (tm) compatible parallel printer port, the other as a general purpose I/O port. These software controlled ports may be used for modem reconfiguration, direct control of an rf deck, controlling status indicators or other purposes.

The HDLC and parallel ports utilize an interrupt method known as Mode 2 on the UP. This allows the interrupting peripheral IC to identify itself to the uP at the time the interrupt is acknowledged, resulting in a very efficient interrupt structure. Further, software ompatibility with the 280 opens the door for many experimentally inclined packeteers to develop, test and refine the software used on the NNC.

In order to encourage such experimentation, a second PC board was developed for the NNC. This board simply plugs onto the NNC digital board and provides a complete, DMA-capable floppy disk controller. Up to four floppy disk drives may be operated by this controller. ZRDOS (tm) or CP/M-80 (tm) operating systems may then be run on the NNC, assuming a proper BIOS and bootstrap ROM are available.

The floppy disk adapter is based on the Standard Microsystems FDC9266 controller, which is an integration of the NEC uPD765 controller with data separator circuitry.

Modems

A third PC board contains a set of four modems as well as a tuning indicator.

The modems are based on the XR2206 and XR2211 FSK modem ICs. These are low-speed (1200 baud default for three, 300 baud for one) modems intended to provide access to the NNC by Amateur stations using standard TNCs. The 300 baud modem is connected to the tuning indicator for ease of use on HF.

Unlike earlier TAPR modems these contain no switched-capacitor input filtering. A passive filtering system is used, resulting in simplified circuitry. The demodulators have been tested in the prototype modem board and a signal of 5 mV peak to peak has been shown to be sufficient for stable capture and lock of the PLL. This high sensitivity is due in part to the use of the XR2211 demodulator IC, in part to the use of a four--layer PC board and in part due to careful selection of modem operating parameters.

The modem board includes a crystal-controlled baud rate generator and each modem includes a state machine to recover clock information from the incoming NRZI data stream. In addition, circuitry is included to convert the NRZ data from the SIOs on the digital board to NRZI for transmitting purposes. A watchdog timer is included in each modem to guard against inadvertent lockup of a packet channel.

The modem board is extensively decoupled to prevent RFI. Similarly, it is designed to be resistant to conducted (incoming) EMI.

General

All boards in the NNC are four layered. This provides an extensive ground plane and power plane for each board, reducing

noise and easing decoupling of the various ICs.

The digital and modem boards conform to the physical form factor outlined by the California WestNet group. These boards are 5.75 by 7.75 inches and include mounting holes to allow them to be attached to the bottom of a standard 5-1/4" floppy disk drive. The power connector is identical to those on such drives, as is the pinout of the power connector.

The floppy adapter card, which piggybacks onto the NNC digital board, has mounting holes which align with those of the digital board when the cards are interconnected. It obtains power directly from the digital board,

The NNC is designed around CMOS circuitry wherever possible. Applying this technology results in wider noise margins, less heat and lower power consumption than other approaches. The net result is increased reliabilty.

Summary and Conclusion

The TAPR NNC is the result of design suggestions from experienced packeteers all over the U.S. It incorporates features making it easy to develop software for, implements a philosophy leading to high reliability and provides a common hardware base tailored specifcally for networking protocols in the Amateur packet environment.

At the beginning of 1985, there were perhaps 2,500 Amateur packet radio stations. At the beginning of 1986, that number had mushroomed to well over 10,000. Hundreds of newcomers are Joining our ranks every month.

Packet channels are becoming clogged to the point of unusability in many areas of the country. Networking is needed to help reduce congestion and improve the overall capability of the burgeoning Amateur packet radio community. The NNC, if adopted by the Amateur Packet community, may provide a solution for development and implementation of Amateur networking protocols.

AUTHENTICATION OF THE PACKET RADIO SWITCH CONTROL LINK

Hal Feinstein, WB3KDU
Amateur Radio Research and Development
PO Drawer 6148
McLean, VA 22106-6148

Abstract

This paper discusses the design of a simple authentication method which is applied to a remotely sited packet radio switch. The control path to the packet radio switch is a very high frequency (VHF) radio channel which is easily monitored and accesed. Such ease of access requires that only authorized control stations be permitted to issue switch control and maintenance commands.

The authentication design discussed in this paper provides three functions: (a) positive identification of the switch and control operator, (b) safeguard message streams flowing between switch and control operator, and (c) rapid identification and rejection of false or manipulated messages.

While some aspects of the work are unique, many of the ideas we employed are disscused in the literature [Mayer,1982], [Needham,1978]. The work on challenge numbers discussed in section three was motivated by a description of World War II spoofing and IFF problems described in [Bethancourt,1979][Kahn,1976] while problems of message alteration are covered by various sources [ANSI,1985]. The slow dictionary attack discussed in section three was originally described by Blake Greenly of Citibank. Lastly, the area of jamming and imitative deception is covered in [Frick, 19451.

This paper is divided into four sections. The first section contains a brief overview of packet radio techniques. The second section discusses assumptions concerning the radio environment in which the packet radio switch operates. This envirnoment is characterized b unreliable radio path as well as occasionaI spoofing and malicious interference. In the third section we discuss the actions of each of the protocola four procedures and the make-up of its data construct. Section three also constains a discussion of the cryptographic considerations upon which this protocol is based. Lastly, in the appendix we describe an experimental one-way cipher based on a random program technique which we hope to incorporate into future versions of the radio packet switch.

1.0 Packet Radio Overview

Packet radio is the extension of packet switching to the radio media. The original experiments were performed by the University of Hawaii and the DoD Adavanced Research Projects Agency. Since that time packet radio networks are finding their way into satellite, police and commercial uses.

Packet radio is a technique for communicating digital information between stations which share a common line-of-sight radio channel. In this respect packet radio is similar to the local area network protocol ETHERNET; but, using radio as the transmission media instead of coaxial cable.

Communications between packet radio stations are governed by a number of design and protocol conventions which are organized into a seven layer model. Lower layers of the model deals with elementary aspects of communications such as how a station shares access to the radio channel or communicates directly with another station. Higher levels of the model detail common procedures for communicating within a network of stations.

A common packet radio station is composed of three components: a personel computer or terminal, a terminal node controller and a low power VHF transmitter/receiver (transceiver). The terminal node controller; which directs the actual packet radio operation; is commonly designed as a small, special purpose computer containing a microprocessor, protocol decoding hardware and various amounts 0f program and working memory.

Within a line-of-sight radio environment, these packet radio stations form a local network were direct station-to-station communications is possible; either directly or with the aid of unconnected packet repeaters. Typical populations range to a hundred or more stations in some large cities with a typical network diameters to about 50 miles.

Regional packet radio systems are linked by a second type of network which spans the line-of-sight user communities. The inter-city network consists of high speed radio links between local user communities linking packet radio switching stations. The packet radio switches provide a access to the local users and also serve as a reliable relay and routing control point for the backbone links.

A packet radio switch is an unmanned, automatic station which provides two functions: first, it provides local access to the high speed inter-city radio network. Second, it acts as a relay and switching point for the high speed inter-city backbone network. Several packet radio switches can provide service to users in a wide geographical area. In this way the inter-city links form a kind of "long distance service" while the local area can be viewed as a call within the same area code.

To control network operations each packet radio switch contains a specialized command link which permits local system operators control over the switches executive functions. The executive functions; which include the operator connection process; is an independent application process running within the switch at ISO layer (application layer) seven. The authentication protocol is designed to function as an adjunt to the operator application and hence, as a part of layer seven.

The switch is designed to reside in locations which are not readily accessible in order to take advantage of high buildings or mountain-top location, making frequent access by service personel difficult. For this reason the switches executive routines provides tools which are quite sophisticated allowing maximum control over the switch.

2. Authentication on the Command Link

Safeguarding the radio packet switch control link is required to prevent unnecessary interruption of service. The common methode of protecting such a link is to use authentication cryptography which offers some unique opportunities to employ various types of cryptographic ideas. The central problem which must be solved is authentication in an open and easily monitored radio channel.

In this section we examine four different assumptions concerning types of attacks against the command link: pervasive monitoring, deliberate interference, playback, and spoofing. These threat assumptions are combined with several

design considerations to yield nine system requirements. In the second part of this section we discuss how each threat is met by a protocol defense or counter-strategy.

2.1 Threat Assumptions and Requirements

The packet radio control link provides the system operator and system developer with a wide range of opertions and testing function. For these reasons it would be quite easy for a malicious operator to disrupt service or "crash" the switch by gaining control of the control operator functions. Short of this, a malicious station could spoof or jam the link, blocking all communications on the link path. Therefore, there are essentially three type of problems to consider: (a) false identification of the control operator or switch, (b) manipulation, insertion or deletion of valid control messages and (c) deliberate interference. We state these as formal assumption below:

1. (Monitoring) It is assumed that a malicious operator is always monitoring the control link and that he has complete knowledge of switch operations.
2. (Playback) It is assumed that the malicious operator can reliably playback valid control sessions or parts of valid control sessions without error.
3. (Deliberate Interference) It is assumed that the malicious; operator can jam the control channel at will either through continuos jamming or through selective "spot" jamming.
4. (Spoofing) It is assumed that the malicious operator can perform selective editing of valid messages.

From these four assumptions nine authentication design requirements were developed. In most cases these represent countermeasures to the attacks described above but some reflect design choices made on the part of the authors.

1. The protocol should gracefully shutdown under error. That is, the protocol should not block a control link by continuos cycling in an error retry or resynchronization loop. (Malicious Interference).
2. Only a valid user can initiate a control session.
3. The authentication technique must not expose key bits on the open air (Monitoring).
4. A unique session key must be used for each new session. (Playback)
5. Critical protocol information must be authenticated (Spoofing).
6. Control message content must be protected (Spoofing).
7. Consequtive control message must be protected. (Spoofing).
8. A control session and its messages must be uniquly identified. (Playback).
9. Control messages must be in plaintext (FCC Regulation).

2.2 Interference Safeguards

The first requirement deals with deliberate interference which forms the most common expected attack upon the switch control link. It is easiest to mount and if done cleverly, one of the hardest to protect against. For purposes of this discussion two types of jamming will be considered: steady jamming which simply blocks the communication channel for a length of time and momentary jamming, in which the jammer induces errors on a regular basis to effectivly block the link.

Steady jamming is essentially a game of strategy in which the jammer attempts to block communications by transmitting signals which the control link receiver cannot tell from the valid signal. To do this the jammer must either overwhelm the switches control link receiver by transmitting a very powerful signal or closly imitate the control link signal characteristics so that the control receiver cannot distinguish the jammer from the valid signal.

To overcome jamming the control link must use special error correction coding and vary some characteristic of his transmitted signal in a fashion which is known only to himself and the link receiver. Military systems commonly employ spread spectrum modulation to provide this unpredictable changing. Simply, spread spectrum consists of either changing the link radio channel many times a second under control of a psuedorandom generator or change the waveform of the transmitter many times a second. In both cases, the link receiver will know what frequency or waveform is being used and can reject other signals not coded in the expected fashion.

For the jammer to be sucessful it must imitate the link as closly as possible; however, the signal is changing so rapidly, and in an unpredictable fashion, that the jammers chances of sucess fall as the links speed of change increases.

In the case of momentary jamming, the jammers strategy is to block communications by attacking the link protocol error detection mechanism. The jammers signal is meant to cause the protocol to perform error retries and hence clog the channel with retransmission. Cleverly applied, this form of jamming need only transmit for a fraction of a second making location by radio direction finding difficult.

Most links subject to deliberate jamming are specially coded with an effect-ive error correcting code (foward error correction) before transmission. The error correcting code can identify and correct many small single bit and burst errors. Commonly, bit rearrangment (bit interleaving) within a data block is also used to combat longer burst errors.

The AX.25 protocol upon which the link is based uses an error detection and retransmission strategy and is vulnerable to both momentary and steady jamming attacks. Foward error correction and spread spectrum hardware are still relativly expensive; hence, a different counter-strategy was needed.

The counter-strategy developed for the switch rests on two observations. First, the switch is designed to always reset and continue automatic packet switching operations in the event of an error or failure. Hence, if jamming interrupts a control operation and denies access over some time period, the switch will resume normal automatic operations.

The second observation concerns safe and unsafe states. Briefly, a safe state is one in which the switch can operate normally and is not in danger of erroneous operations. Comparatively, an unsafe state puts the switch in danger of crashing, perhaps through a temporary instruction patch or experimental manipulation of parameters.

Commands issued by the control operator affect the state of the-switch. These operator commands are issued individually or as part of a group of commands. Moreover, each intended operator action ends with the switch in a safe state, while commands within a sequence of operator commands may place the switch temporarily in an unsafe state.

The most dangerous time for the jammer to become active is when the switch is in an unsafe state. We assume that the jammer is monitoring the activity of the control link and can determine when this state occurs. To close the window of vulnerability the switches executive software sets a hardware timer whenever the switch enters an unsafe state. If the link should fail for any reason this timer will expire and trigger a hardware reset returning the switch to normal packet switching operations.

We have choosen not to employ special coding and spread spectrum hardware primarily as a trade off between the effects of jamming, jamming frequency and hardware expense. Jamming which occurs for a period of one second or more can be easily located with current., commercially available automatic radio direction finding techniques and specific legal remedies can be applied. Jamming attacks; while distruptive; are generally not too frequent; however, in certain specific areas of the country jamming is more prevalent and it may be necessary to employ some of the anti-jam hardware mentioned.

2.3 Imitative Deception and Spoofing

Imitative deception and its variations form the basis for several attacks. The ones considered here are attacks whose strategy is to imitate a vaild user action by manufacturing valid looking messages, altering selected parts of a valid message (spoofing) or by recording and replaying whole or parts of a previous session (playback attacks).

The strategy of imitative deception is to inject a valid appearing messages into a link which is taken for valid traffic. One of the primary [Frick,1945] instances of this type of attack occured in the opening days of WW I on the German-Russian front durring the battle of Tannenburg. In this battle the Germans; using the call signs and radio procedures of the Russian High Command; were able to send false instructions to various commanders countermanding previous orders and altering battle plans. This resulted in a military disaster for the Russians with far reaching consequences.

The major safeguard against imitative deception is an authentication procedure which can guarantee that each message has originated from the authorized user. Communications systems commonly rely on cryptographic authentication procedures which require the user to perform some enciphering operation with a known key. The system performs the same operation in parallel and compares its result with the potentical users. If the results are identical, it is assumed the user possesses the proper key and hence is an authorized user.

The authentication protocol described in this paper uses a parallel encipher and test procedure to safeguard three aspects of the control session: establishment, message flow and exception handling. Each of these three aspects of control link communications offer the spoofer an oppurtunity to mount an attack against the system.

The first safeguard is placed at the point where a user requests the switch to begin an operator dialog. The user would up to this time have established at least an AX.25 level connection to the switch and request to connect to the switches executive software. In some respects, the connection request resembles a conventional computer logon with a secret password; however, the radio channel is always being monitored and the password would not stay secret very long.

The mechanism used by the protocol relies on parallel encipherment of a known constant by both the user and switch. A challenge number (CN) is generated by the switch based on a randomized timer value and is transmitted to the user in response to his connection request. Both the user and the switch then encipher the challenge number using a previously distributed secret key. The user returns the enciphered value to the switch which then performs a comparision with its locally calculated value yielding a match for a valid user.

If the returned value does not match the switches enciphered value it is assumed the user does not possess the secet key. The connection request is denied and the switch breaks the AX.25 connect ion ("hangs up"). At this point no new operator dialog connection requests will be accepted for a period of fifteen seconds. The goal is to prevent a brute force attack by microcomputer which could rapidly test many trial keys.

A benefit of the randomly generated challenge number is that it provides a unique value to associate with each operator connection. This unique value will be used to differentiate messages in the operator session from those of an old session and hence block a playback attack. It is important to note that the switch and not the user generated the challenge number. If the user could generate the challenge number, a spoofer could simply playback an old challenge number and setup to replay the associated old session.

Having been blocked from directly impersonating a valid control operator an attack now open to the spoofer is to attempt to disrupt the control link by insertion,,modification or deletion of valid control link messages. Neither the link nor network protocols offer protection against inserted messages [Borden,1985]. In fact their action is to accept the first correctly numbered packet and ignore subsequent packets with the same send and receive counters; --accepting the spoofer packet while ignoring the valid one.

To overcome this difficulty we have included a message sequence number as part of the authentication construct. The authentication protocol tracks the sequence numbers by computing the next expected sequence after each valid message is received. This value is saved by both the switch and the control operator and is used to compute the next expected message authentication value.

Message alteration is an attack open to the sophisticated spoofer. The goal of the spoofer is to intercept and use a valid frame and packet structure but insert a spoof message into the I field portion of the packet. This attack is more common on wire line systems were an attacker need only insert a cornputer in the line to perform the alteration. In the radio environment, an attack of this type is still possible if a spoofer could automatically intercept: a valid packet, insert the spoof text and retransit the packet in under a few seconds.

A senerio of this attack might be for a spoofer to intercept the incoming packet from the control station by aiming a highly directional antenna at the control station while a second spoofer, closer to the packet switch momentarily blocks reception. This can be done by transmitting a short noise burst to jam the packet switch control receiver making the switch unaware that a valid packet was sent. After intercepting the original valid packet, the first spoofer would overlay the packet I field with the command of interest, recompute the frame check error value and then quickly re-transmit the message to the packet switch.

To defeat this attack the authentication protocol contains a modification detection indicator (MDI) which detects any modifications to the message text. Various types of check-sums have been studied for this purpose and certain vunerablilities have been identified where the "sum" is composed of linear operations. In the authentication protocol used by the packet switch a nonlinear approach is used to reduce this types of exposures.

The MDI value is computed over the message contents including the authentication state indicator of the authentication protocol construct which prevents spoofing. The authentication value is then computed by enciphering the combination of MDI value with both the Challenge and internally stored message sequence number.

3.0 Authentication Protocol

The authentication protocol is essentially a computer oriented protocol with the control operator executing part of the protocol on a personel computer and the remainder executing within the switch. The two communicants exchange an authentication data unit (ADU) which is affixed to each operator message. In the return direction an ADU is used to signal acknowledgments and special conditions and can be received independent of a switch message. Contained within the ADU is the authentication state indicator (ASI) which signals conditions between the two ends.

Figure la. Control Message Format

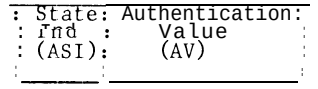


Figure lb. ADU Fields

Figure la shows the operator control message and ADU layout. The common control message typically takes the form of an operator command such as instructions to check status, start or stop operations and on-line debugging. The actual contents of the message can take either a conversional English format or a machine readable format depending upon where the actual command parsing is performed. In the packet radio switch under discussion, command parsing is done at the control operator location and so only machine readable information is exchanged.

Figure lb shows the format of the ADU. The authentication state indicator (ASI) is used to signal the state of authentication and indicates status such as acknowlegements or a command such as session termination.

There is an associated message count field which is stored internal to the switch and the control operator. The count field is used to track the message number and detect lost,inserted or duplicate message. This field is 16 bits in length which was judged adequate for even exceptionally long control sessions.

3.1 Authentication Protocol Description

The authentication protocol consists of four procedures: authentication establishment, message transfer, terminate and error. The four procedures of the protocol operate in series with the exception of the error procedure. This procedure is charged with recovery when a message fails to authenticate. In the authentication establishment procedure the control operator is attempting to connect with the packet radio switch. The procedure calls for the control operator to establish a lower level connection with the switch operator control routines. Once this is done a reliable path is assumed to exist between the control operator and the packet radio switch.

The next step is for the control operator to authenticate himself to the switch by properly encrypting a challenge number (CN) under an initial key derived from the master key and the challenge number (the challenge number serves as a key indicator function). The switch generates the CN via a clock driven random number generator and sends this value to the control operator. The control operator now enciphers the CN by combining it with this key and sends it back to the switch over the control link.

In parallel with the control operator, the switch has also enciphered the CN value for comparision with the one received from the control operator. A match! indicates that the control operator is in possession of the same initial key thus completing the authentication initialization phase.

To add security, the validated CN value is combined with a constant and re-enciphered to produce a session randomizer value (SR) which is stored internally by both the switch and the control operator. This step limits the usefulness to a spoofing station of a intercepted CN value since it will not be used futher in the current form within the authentication procedure. An important purpose of the SR is to prevent playback of a previous session. The SR associates with each session serves as a randomly selected session identification number. The SR is also cne of the fields used to construct the message authentication value for each operator message;, making this value reflect the individual session SR.

Two different approaches were considered in generating the SR value. The first approach is to consider a monotonically increasing counter value to provide unique session numbering. The drawback with the counter approach is it requires long term storage of the counter value. This conflicts with one 0f the switches key design principles of never relying on values which could be modified by a software error for critical operations. Typically, constants and routines are comitted to ROM for reliability; comparatively, values contained in RAM are subject to both software error and system resets.

The second approach; which is used in this implementation; is a SR value produced by a twice enciphered clock driven random number generator. The first encipherment producing the CN value and the second yielding the SR. This approach produces evenly distributed random numbers which have a very small but non-zero probability of repetition, but which needs no long term counter storage.

When an initialization attempt fails, the switch returns an ADU with the authentication reject flag set. At this point there is an oppurtunity for a dictionary 'attack if the switch allowed instant reconnection. Instead. the switch does the equivlent of "hanging up" by signalling the lower level protocols to break the X.25 connection to the control operator and then enforces a 15 second wait before it accepts any new authentication establish requests. This action blocks the ability to qucikly retry successive test keys; one of the prime elements necessary for an economical dictionary attack.

One additional style of dictionary attack should be mentioned. This style of dictionary attack submits test keys intermittently over a long period of time. The conventional dictionary attack relies on speed to submit many test keys in a short time. This style might be termed the fast dictionary attack; comparatively, a slow dictionary attack spreads its tests out over a long time in order not to arouse the suspicions of a control operator. The slow dictionary attack relies on the fact that the system is available around the clock so that tests need only be conducted sporadically.

The goal of the slow style of dictionary attack is; like the fast attack; to find the key currently being used. The slow dictionary attack may be launched when breaking the system is of low priority sort of a bonus if it could be done. It is difficult to guard against this attack since few attempts would occur over an interval. The obvious defense is to use a very large key space to forces the attacker to maintain a large dictionary and perhaps close the time window during which the attacker might test. For example: allow connection requests only during business hours. This strategy is similar to what was done durring World War II with the identification friend or f'oe (IFF) systems. The enemy would test the bombers IFF by sending trial messages hoping to find the current settings. To limit this, the IFF boxes were commonly switched off until within a range where they were needed. This deprived the enemy of their window.

The second procedure of the authentication protocol is the message transfer procedure which authenticates each arriving message. The actual authentication value (AV) is generated by:

$$AV = \text{ENCRYPT}(\text{Key}, \text{MDI}(\text{message-text}) + \text{SR} + \text{Sequence-number})$$

The AV itself is 64 bits in length while the modification detection indicator is a 64 bit value (non-secret) which is calculated on the state indicator. The ADU will be affixed to the end of the current message and transmitted. When the switch receives this message, it will recompute the AV using the expected next message sequence value; and the SR; both of which are retained independently of the control operator. The switch will also recompute the MDI on the received message and derive a test AV value.

Authentication of the current message is performed b comparision of the test AV with the received A%.

Once authentication has taken place there are three possible actions: pass authentication, fail or fail on retry. Messages which pass authentication cause the message counter to be incremented and the message to be passed to the switches operator function. If the message fails to authenticate then no message will be passed to the operator function, instead an error procedure is entered.

In the error procedure, the counters are not updated to prevent a counter synchronization problem which would complicate things later. Instead, the message as received is sent back to the control operator with the reject flag set. The control operator can then retry seven times before the switch declares a formal authentication failure. When such a failure is declared, the authenticated session is cancelled, the switch "hangs up" and the unconnected state is re-entered.

Finally, the termination procedure is used to gracefully shutdown the authentication mechansim. Termination of an authenticated session is always initiated by the packet switch in response to a disconnect command, software failure or idle terminal timeout.

3.2 Cryptographic Considerations

There are a number of design issues surronding the choice of cryptography-applied to this authentication problem. In this section we examine several of these issues including legal, technical and operations and, identify why specific choices were made.

A legal requirement which steered the choice to authentication cryptography rather than-message encryption; which offeres alternative solution techniques; is the plaintext requirement for message information. The plaintext requirement is a regulation which the Federal Communication Commission (FCC) has levied on different communications services, generally prohibiting them from transmiting enciphered information. Until quite recently this meant any type of cryptographic or "scrambling" technique.

When the spectre of unauthroized use of control links such as satellite command channels began to attract serious attention, the FCC relaxed its strict interpertation to permit some use of authentication cryptography. The general test is if the specific application enhances or facilitates communications as opposed to masking its meaning.

With the relaxed definition it has been possible to utilize an on-the-air control link. Commonly, in the past it was necessary to utilize either a dialed or leased telephone line for control of a repeater to solve the operator authentication problem. The expense of installation and maintance of telephone lines; specifically to inaccessible places such as mountain tops; often contributed to the choice of switch placement.

3.3 Implementation Issues

The current system is based on private key ideas allowing us to employ DES, a commonly available commercial cipher algorithm. Three factors drove this choice: economic considerations, cryptographic strength and commercial acceptability.

Economically, DES is available in various hardware formulations and hence, including it at a low cost was possible. We chose a hardware formulation because in terms of storage, corn utational power and delay a software implementation would not be advantageous. A second reason was the large outlay in time and expense for the software assurances necessary for cryptographic processing.

Software assurance techniques for cryptographic systems are concerned with prevening information compromise due to software failure. A well implemented cryptograpic system commonly seperates the secure and non-security aspects of the system and in addition, requires the software design to go through a rigorous failure mode analysis identifying each failure mode and correspondin g assurance. The increased reliability afforded by this procedure guards againt failures in which the cryptographic process fails to encipher, garbles the information in a reversible way or worse, allows key bits to erroneously appear on the communications channel.

In choosing the DES algorithm we surveyed the relavent literature on its cryptographic strength and have found no clear example of a succesful attack. We find the arguments against DES fall into three classes: (a) super-computers have put a known plaintext attack within reach, (b) the government already posses such computers (c) there exists a secret process (a trapdoor) that reduces or negates the computational burden required in solving DES.

In considering each of these three classes of aurguments only the first seems to be clearly verifiable, implying that a careful re-examination of DES will be needed in the future. In the second class; who currently possess such computering power; is largly irrelavent to the switches threat environment. Lastly, the existance of a trapdoor procedure could jepordize the security of this authentication procedure. No clear indications of such a trapdoor have been found in the literature, but if such an attack should come to light DES would have to be replaced.

3.4 Form of DES Utilization

The cryptographic aspects of this protocol are based on parallel encipherment; by both the switch and control operator, of non-secret information under the control of a unique secret session key. With the exception of the key all information is either broadcast or easily calculated and hence an attacker could assemble a sizable collection of plaintext-ciphertext pairs for study. The goal of such a study is to deduce the key value by a close examination of the enciphering process. Therefore, algorithms selection must be limited to those which can resist a known plaintext attack.

3.5 Key Management

A simple key management scheme was developed for the switch to increase the period between rekeying (the cryptoperiod). The technique also increases security by creating a unique key for each session thereby limiting the amount of message traffic under the same key. The scheme implemented for the switch employs a fixed master key of 512 bits which is shared by both the switch and control operator.

The switch generates a session key after the authentication establishment procedure is sucessfully performed by computing a privatly held function on the CN. The output is then used to select 56 bits for the current session key. Since each session uses a different schedule of key bits from the master key, the rate at which an attacker could infer the next session key would be very low.

4.0 Conclusion

This paper has decribed an authentication procedure to control a remotly sited packet radio switching node. The chief problem to be solved was to provide a authentication technique which operates in the face of occasional jamming, spoofing and prevasive monitoring.

The protocol is designed to authenticate a single direction of message flow -- from the control operator to the packet radio switch. Bi-directional and full duplex extensions of the protocol are possible; however, there are special complexities, chiefly in the area of counter and error management which must be solved.

Finally, there is interest in extending the design of the authentication protocol to address new and wider problems within the packet radio network. Among these are extension to the full duplex case, authentication of the switch-to-

switch control protocol and multiple operator situations in which there is a need for split authority. Public key techniques seem to offer many interesting approaches and will probably be used in futher protocol versions.

Appendix

An alternative One-Way Enciphering Function

In the commercial setting in which the packet radio switch is to operate, DES was choosen and implemented as a hardware accessary to the main switch microprocessor. To utilize the DES as a one-way function, the key was first combined with the value to be enciphered and then the result is used as both the message and the key. The output from the encryption operation is taken as the result of a one-way transformation.

While DES was choosen for reasons of its commercial acceptability, test versons of the switch employ a previously described one-way function based on a "random program" (RP) technique first described by Evans, Kantrowitz and Weiss of MIT [Evans,1974]. The goal of developing a new one way algorithm is one of research since one of the one-way cipher described in the literature would suffice for a DES alternative. In as much as the RP one-way function is strictly experimental it could not be offered in place of DES, yet in terms of efficency and compactness it seems superior at this point.

A one-way function is relativly efficent to compute (encipher) but computationally infeasible to invert. In most cases one-way functions are based on a known hard problems choosen from complexity or number theory such as the knapsack, prime factorability or root extraction.

The random program (PR) algorithm is not based on one of the accepted hard problems but on the execution of a series of machine language instructions whose order depends on the specific contents of the algorithm argument. Abstractly, the function consists of a register (R), which contains the value to be enciphered; a set of machine language instructions (M), whose operations alter the contents of the register and a selector function (S) which specifies the machine language instruction to be executed next.

Roughly, the algorithm is executed by the selector function taking a selection of bits from the register and computes an index which designates one of the machine language instructions in M. The designated instruction is then executed, altering the contents of the register. This cycle is repeated some number of times and yields an output value in R.

In order to invert this algorithm the analyst would have to know what order the instructions in M were executed. This in turn depends on the contents of the original value to be enciphered which is unavailable after the algorithm completes execution.

Three factors make the RP function attractive; first, the function does not rely on extended precision arithmetic commonly seen in many of the public key approaches. Second, the computation and storage requirements are competitive if not better than conventional approaches, an important element in the switch design and lastly, the low complexity of the algorithm leads to simpler implementation.

The "pure" RP algorithm described above has a number of weaknesses which must be addressed before an solid implementation could be offered. In the RP algorithm the cryptanalytic strength is derived from the unpredictability of the order of instruction execution. Essentailly, each argument to be enciphered should select a unique series of instructions with each instruction given equal chance of execution at all times.

Both the selector function and the value to be enciphered contribute to the specific order of execution. The selector function output must therefore have a flat probability distribution so that all choices are equally likly over the range of values to be enciphered. Since this range is composed of all values which can be held in the register, all representable bit patterns must be expected.

A simple selector function could be heavily influenced by the bit patterns in the input. This inturn would reflected in the order of instruction selection and execution. The selector function therefore must attempt to "whiten" the values selected from the register and hopefully limit the vulnerability from patterns in the input.

A second area of concern is degeneration in the radomness of intermediate values. For example if the selector function operators on the results of each instruction execution, it is quite possible for patterns to appear which tended to "converge" either by alternating or by containing less and less variety. In either case the equiprobable instruction choice would be compromised.

The chief disadvantage of the RP algorithm is the lack of in-depth understanding of its cryptanalystic strengths and weaknesses. Currently, it is undergoing detailed study and analysis at the end of which we hope to submit the results to peer scrutiny.

References.

ANSI 1985, "X9.9 Financial Institution Message Authentication". American National Standards Institute, New York

Bethancourt, T.E., 1979, "Instruments of Darkness", Holiday House, New York

Borden,D,Fox,T. 1985, "Spoof Resistance of HDLC,X.25 and Certain Commercial Transport Protocols", Private Communications

Diffie, W and Hellman, M. 1976, "New Directions in Cryptography". IEEE Trasactions on Information Theory IT-22, 16(Nov).

Evans, Rantrowitz and Weiss, 1974, "A User Authentication Scheme Not Requiring Secrecy in the Computer", CACM, Jan 1974

Frick, 0. 1945 "War Secrets in the Ether", Aegean Park Press, Laguana Hills, CA

Kahn,D. 1967 "The Code Breakers, The Story of SecretWriting". Macmillian,New York

Konheim, A. G. 1981 "Cryptography, a Primer"', John Wiley and Sons, New York

Meyer,C.and Matyas,S. 1982 "Cryptography". John Wiley and Sons, New York

Needham,R.M. and Schroeder, M.D., 1978 "Using Encryption for Authentication in Large Computer Networks". Comm of ACM 21, 12(Dec)

Newland, P. 1984, "A Few Thoughts on User Verification Within A Party-Line Network" Proc. of The Fourth ARRL Computer Networking Conference, American Radio Relay League, Newington, CT

PACKET SWITCH SOFTWARE DESIGN CONSIDERATIONS

Terry Fox, WB4JFI
President, AMRAD
1819 Anderson Rd.
Falls Church, VA 22043

Abstract

Toward the end of 1985, a new device showed up on Amateur Packet Radio, the packet switch. Soon, the packet switch will be replacing digipeaters around the country, giving more reliable (if slower) operation of the overall network. The first switches have been based on TAPR TNC-2 hardware, and therefore are somewhat limited. An accompanying paper describes what types of hardware the author sees being used in the future (both near and down the road) for the switches. This paper will explain the author's view of how the switch software should be organized.

This paper will not provide actual switch code, but rather indicate where progress is being made, and where help is needed. Also, wherever possible, I cite Protocols and Standards I believe should be implemented.

Types Of Packet Switches

Before we delve into the meat of the software inside the switch, we must decide what the switch's function(s) are. I see these functions falling into two catagories, with one having two sub-catagories; the transit switch, and the end-point switch.

The first type is called the transit switch because it is used to pass network connections through it, but not necessarily act as an end-point of a network connection, except for maintenance functions. The transit switch can be thought of as a software "patch-panel" type of device. This means that packets coming in from one adjacent switch station will pass through the transit switch to another adjacent switch. The length of time the "patch cord" is plugged in, allowing the connection, depends on the type of networking protocols being used. It can vary from a very short period (the single packet) in a dynamic-routing datagram network, to almost forever in a permanent virtual circuit case. Normally, the patch is maintained for the length of the end-to-end Network connection in virtual circuit networks. The transit switch may also perform a similar function as today's remote location digipeaters, that of interconnecting local networks to form a larger network.

Transit switches may have special hardware needs, since they will most likely be placed at remote locations far away from network users. Among these hardware needs will be low-power operation and redundancy of key parts for survivability. Additional software may be required to take advantage of this hardware.

The other type of packet switch is the end-point switch. While it may also perform the transit switch duties mentioned above, it will also have software that allows the end-users of connections to interface to it. This is where a typica1 Amateur station will put his packets "into the pipe":) and also where the packets come "out of the pipe" at the other end. In our patch panel scenario mentioned above, the end-point switch is analagous to the last patch panel that has the equipment connected to it. In order to perserve the capability of using older packet systems that may be capable of only level 2 connections, there are potential1y two different types of network interfaces in the end-point packet switch.

If the end-user is an Amateur that has a newer packet board (called a PAD) with true networking code in it, Amateur A can ask for a Network Layer connection from the switch. The switch, understanding that a PAD is requesting connection, will automatically provide the initial link and network connections, and proceed to attempt to place the requested call to the destination station.

If, however, the end-user is an Amateur using a TAPR TNC-1 with level 2 (link) only software, there must be some additional method of providing a network interface for the user. Typically, this will be through an additional program, which I will call the network access PAD routine. This network access PAD routine will run as an application layer program inside the end-point switch, and will provide all the necessary network interface magic for the Link-only Amateur.

Let me now describe how I think the switch software might be organized.

Top-Down Design of Switch Software

One reason for this paper is to indicate how we in AMRAD are working on the switch software. Up to now, most packet systems have been designed by first putting together some hardware, then writing low-level drivers for that hardware, then writing higher level code to properly tweak the low level drivers, etc. While this building-block approach does indeed work, it can lead to problems if some unanticipated problems occur. It also tends to be difficult for more than one person to work on a project using this approach. AMRAD has started working on the packet switch using more of a top-down design approach. While what we are doing may not be exactly top-down, it does follow that philosophy.

We are studying the various aspects of the switch BEFORE writing a line of code. This includes how the various processes inside the switch interoperate, and how to make the best use of the hardware supplied.

As part of this study, we quickly came to the conclusion that in order for the different software "machines" of the switch to work together properly, some sort of operating system should be implemented. This operating system need not be as complicated as a disk operating system, but should provide the necessary tools for the different processes to intercommunicate. It should also provide a common set of utilities for access to shared switch resources, such as the memory or buffer pool, and timer subroutines. I will pursue the operating system in more detail shortly.

We decided to follow the IS0 Open Systems Interconnection Reference Model (OSI-RM) when splitting up the task of designing our switch. This gave us some rather clear divisions of responsibility, both for dividing our labor pool for the design work, and for what software is responsible for accomplishing which sub-task of the overall switch.

I feel that most sub-tasks should have two distinct parts; the operational code, and a management section that errors are reported to and new requests of that sub-task are coordinated through. This will lead to more orderly code at each sub-task, and also provide a better form of sub-task intercommunications.

Some of the sub-tasks of a typical Amateur packet switch are as follows:

A. Operating System (including resource management).
B. Maintenance Application of switch.
C. Message Authentication for maintenance.
D. Database management (User and Routing).
E. Network Access PAD Application (if available).
F. Session Layer Protocol for internal switch applications.
G. Transport Layer Protocol Machine.
H. Network Layer Gateway Machine (if needed).
I. Network Layer Addressing and Routing routines.
J. Network Layer Protocol Machine.
K. Link Layer Protocol Machine.
L. Physical Layer Software Support.

Each of these sub-tasks will now be discussed individually.

Operating System

A typical packet switch will need to perform several different processes, seemingly at once. One method of accomplishing this would be to use a separate microprocessor for each sub-task. While this might make the writing of software easier, it is generally considered a waste of processing power, costs muchmore, and takes more room and power. Until this method of dividing the overall switch task becomes necessary (due to single processor overload), another method can be used more effectively. Someday, as RF channel speeds go up and as more RF channels are used per switch, the use of multi-processors will become necessary.

One microprocessor should be able to perform all the necessary functions of today's packet switch if some method of dividing up it's processing time is devised. This division of time can be done in many ways. One of the most common methods is to have the hardware provide a timed interrupt, which the processor uses to indicate it's time to switch tasks. This way, each task is allowed a certain time slice to perform as much of its duties as possible. Another method of dividing the processor time is to let each task run to completion before switching to another task. There are also various combinations of these two task switching methods. The method of switching, along with the necessary support required to make this task switching transparent to the tasks is one responsibility of an operating system.

Almost all processes in the packet switch will need to communicate with at least two other processes, both to pass information and to request services from the other process. The traditional method to do this is to call a subroutine (or function), passing a pointer to the information to be tranferred, or the service request. Each process may be written at a different time, or by a different person. This sometimes leads to different interprocess interfaces, making the overall system design more complicated. A standard, predefined set of routines to provide information passing and service requests would aid in the designing of the switch. This is another service that is often provided by an operating system.

Still another service that would be frequently requested would be a common set of subroutines to provide memory and buffer management. The packet switch is basically a message transfer machine, with the messages being the packets. These messages are usually maintained inside the switch in buffers. The requesting for, and the freeing of buffers, along with the passing of buffers and accessing of data within the buffers, can easily be standardized into a common set of subroutines. This set of subroutines lumped together is called buffer management. Buffers are made from computer memory, and this memory is usually controlled by another set of subroutines, called memory management. Since buffer and memory management routines are services used by almost all processes, they are normally considered part of the operating system.

The packet switch has several processes that rely on the use of timers for error detection and recovery. In more advanced systems, such as the packet switch, software timing loops are not considered good form, since they occupy too much processor time, essentially putting to sleep all other processes just to keep time. The suggested alternative is to use hardware timers with software support for those timers. Since timers are requested by various processes, it makes sense to provide timer access as an additional service by the operating system.

Noting that the use of an operating system that provided the above support would be beneficial both to those of us writing the switch code now and to those writing additional code for the switch or modifying our switch code, we started looking at available operating systems. Our prerequisites included that the operating system be either free or inexpensive, preferrably written in a higher level language for software portability, and be efficient at its appointed jobs.

Our present packet switch hardware is based on Intel microprocessors (the 8080 and 8088 families), at first it looked like iRMX from Intel might be a good start. It had all the necessary support capability mentioned above, along with a lot more. Upon careful analysis however, it began to look like iRMX would take too long to perform most of the requested tasks.

About the same time we decided iRMX wouldn't fill the bill, Mike O'Dell started talking up a different type of operating system, called the HUB. It was designed to be more efficient in message based applications, especially in packet type devices. The more he talked, the better it sounded. HUB does not have a lot of fancy stuff, such as processes interrupting other processes, but for our application we don't need that fancy stuff, which nine times out of ten gets in the way and takes valuable processing time. Mike was so convinced that the HUB was the right way to go that he wrote the code for us in C. I have been able to compile his C code on an IBM-PC and on a Xerox 820. We are going to use the HUB operating system as the basis for our packet switch, as soon as we get a more thorough understanding of it. Mike has written an introductory paper on the HUB which is elsewhere in these proceedings. I feel this HUB will become a very important part of our packet switches, since it provides a stable, working interface for the rest of the tasks to use.

Eventually, the operating system should probably be written in assembler, since it will be the most commonly used software in the switch.

Maintenance Application Task in the Switch

Now that we have an operating system running in the switch, there must be some tasks that the switch needs to perform. One task might be the maintenance of the switch itself. This task could be divided into two sub-tasks; minor "tweaking" of variables for more efficient switch operation, and recovery after some type of malfunction. While the various protocols might be capable of doing minor tweaking of their own variables, both of these sub-tasks should be available to an Amateur remote from the switch. This means an Amateur could establish a connection to the switch, indicate the purpose is to perform remote maintenance of the switch, pass through some authorization door, and then perform the maintenance of the switch, as long as the switch is operating.

Among the maintenance functions that should be available are:

A. Take the switch completely off the air.
B. Re-boot the switch from mass-storage.
C. Upload new switch code or portion of switch code over the packet channel.
D. List operating parameters of the tasks.
E. List status 0f connections to switch.
F. List stations and switches heard lately.
G. Gain access to User and Routing databases for listing or updating.
H. Modify operating parameters of switch.
I. Monitor and report switch operation.
J. Dump error reports since last maintenance connection.

If a switch starts malfunctioning, there should be a way to gain access to it, assuming it has not totally failed. Each Physical channel connected to the switch should be allowed to have a connection established through it to gain access to the Maintenance Application program. This way, if one Physical channel is tied up or broken, it could be turned off without removing the whole switch from the network. If however, the switch has some major flaw that prevents proper long-term operation it may be better to totally disable the whole switch until on-site repairs can be performed.

There also needs to be a method of either uploading new switch code, or updating a database from a remote location. The best way to do this is to store the uploaded information on a mass-storage device, then re-boot the switch after the upload has been successfully completed. Modifying portions of the switch code "on-the-fly" could prove disasterous, and should be avoided.

Another maintenance function would be to monitor switch operation. This can be done in real-time (establish a maintenance connection, then let the Maintenance Application program periodically dump status information) or by accumulating the status information in variables or a database for later retrieval.

The maintenance process described above assumes there is not a separate Service Processor connected to the switch, or the Service Processor is not available for some reason. If there is a Service Processor available, most maintenance functions will probably be performed through it, in addition to others that the actual switch could not perform and still operate normally.

Remote maintenance of switches will be a necessary function that will be improved upon as a history of switch operation is documented. There is no way we can predetermine all the maintenance requirements before some actual operating time is logged. This fine-tuning of switch maintenance will be especially important for remotely placed switches, where access may be difficult or impossible for long periods of time.

Since the maintenance application does not have any major speed constraints, it could be written in a higher level language such as C for portability.

The Maintenance Application process will receive its data from one of two sources; a local console if the maintenance is being dons on-site, or from the Message Authentication process if the maintenance requests are coming over a packet channel. The Maintenance Application process should be able request status of each switch process, and control certain aspects of each switch process. The access to each process will normally be through a management routine within that process.

Message Authentication for Maintenance

Since the maintenance application involves direct control of the packet switch internals, some method of allowing certain Amateurs access while preventing other Amateurs access must be employed. Access control schemes such as simple passwords or control codes are not adequate, since any Amateur listening to the packet channel can gain knowledge of them. Since packet radio is a digital communications method, a way of dynamically digitally encoding both the access request and the actual commands sent to the switch can be easily accomplished. Paul Newland presented a paper at the 1985 ARRL Computer Networking Conference describing such a scheme. Coincidentally, Hal Feinstein came up with a very similar scheme at about the same time. Hal has been working on this over the last year, and has written a paper on his activities, found elsewhere in these proceedings. He is also writing the code to perform the message authentication he describes.

The Message Authentication process does not require real fast execution time, so it could be written in a higher level language, such as C, for portability between different types of switches.

The Message Authentication process interfaces between the Maintenance Application process, and the Session Layer Protocol machine, or the Transport Layer Protocol machine if there is no Session Layer.

Database Management

Packet switches will need to know where to route the packets they receive. This routing will be based on information the switch maintains on how the network is organized. Since this information should be quickly accessable, an orderly method of storing it should be used. Also, routing information will change periodically, due to switches going up and down. In order to keep the routing information accurate, a small database manager should be used.

Another function a database manager could provide for end-point switches is to keep a list of Amateur stations it normally services. This way, when a request for one of its users comes in, it will recognize that it should respond, or if the user has indicated that all calls for it should be forwarded to another switch, the switch can pass the new destination end-point switch address back to the source end-point switch.

This database manager does not need to be as sophisticated as a commercial program. A relatively small database program can support these two databases, along with any other databases that might be used by the switch (such as maintenance records). The database program could be written in a higher level language, such as C with no ill effects.

Network Access PAD Application

As mentioned earlier, end-point switches will need to provide two different network access methods to the Amateur. If the Amateur has Network Layer capability it can request connections directly through the switch. If the Amateur only has Link Layer connection capability (such as most present TNC boards have), the switch will have to perform some additional functions for the Amateur.

One method of providing this service would be whenever a Level 2 only TNC requests connection to the switch (note the connection is TO the switch, not thru it as a digipeater is presently used), the switch would accept the Link Layer connection. This Link only connection indicates that the user cannot support network protocols, and the switch then places a request to the Network Access PAD code for assistance. The Network Access PAD routine might then send the user a menu of functions that the net access PAD code can provide, such as:

1. Connect to another Local Amateur.
2. List All Local Amateurs on this Switch.
3. Look up a Remote Amateur's Address.
4. Calculate path to Renote Amateur.
5. Connect to Remote Amateur via supplied path.
6. Connect to Remote Amateur via path implied by supplying Destination Switch Address.
7. Connect to Remote Switch to Monitor it's the remote channel.
8. Add this Amateur station to Switch User directory.
9. Delete this Amateur station from Switch User directory.
10. Indicate an alternate path for this Amateur station (station will be mobile).

The above list is-not meant to be a final version of a menu, but does indicate some of the functions the network access PAD routine should provide.

The user then selects the function or functions to be performed, and the switch takes care of doing the actual work. If for example, the user requests a network connection to another Amateur on the same switch with Network Layer capability. The switch will then request a network connection to the destination Amateur on behalf of the first Amateur. If the network connection is successful, the switch will handle the Network Layer protocol machine for the

Amateur, using the Link Layer connection to provide data integrity between itself and the first Amateur.

One point here is that a lot of additional work will be done inside the switch to provide this service to Link-only Amateurs. If two Link-only Amateurs wish to communicate through a switch, it may be better if they connect directly to each other, using the packet switch in Level 2 digipeater mode. Digipeating is generally not a good idea once true networking arrives, but in some cases it may be the most efficient method of communicating. A link using only one digipeater should be stable enough to provide reliable communications, and the loss of efficiency due to digipeating may be made up by the reduced amount of overhead the Network Layer would otherwise add.

One type of Network Access PAD interface will be based on the CCITT X.28/X.29/X.3 set of standards, commonly referred to as the triplex protocols. X.28 defines the user-to-network PAD interface, X.29 defines the network-to-PAD interface, and X.3 defines the variables used in the PAD, along with some common settings of these variables.

Dave Borden has written a paper on the User interface which can be found elsewhere in these proceedings. The user interface is another task that can be written in a higher level language such as C.

Session Layer Protocol for Internal Switch Operations

The Session Layer is used to multiplex more than one data stream to the upper layers over a single transport/network connection. This may be necessary in the switch for maintenance and for Link Layer only connections. An example of the latter would be if a Link Only TNC requests more than one network connection, or if the TNC requests both a network connection and some other Network Access PAD function simultaneously.

I am leaning toward the use of a sub-set of the CCITT X.225 Session Layer protocol for starters. This protocol is more than adequate for use by the packet switch. The use of a higher level language for the Session Layer Protocol in the switch would not pose any major problems.

Transport Layer Protocol Machine

The Transport Layer is responsible for absolute data integrity across the network connections. While the Network Layer makes the individual connections between switches, the Transport Layer provides a logical connection between the two end-points (source and destination). Different Network Layer Protocols place different requirements on the Transport Layer Protocol.

A network made up of datagram type switches will need a more complicated Transport Layer Protocol machine, partially because datagram switches do not maintain a "connection" between each other, and therefore do not have ANY error recovery (that's recovery, not detection) procedures operating between themselves. Also, the only real recourse a datagram switch has when detecting an error is to throw the offending packet in the garbage queue and hope it is retransmitted.

A network based on virtual-circuit type switches will need little, if any, Transport Layer Protocol machine. Since individual logical connections are maintained between switches involved in a network connection, these individual logical connections will detect and correct almost all errors incurred along the network connection. The only two error conditions that the individual logical connections between switches won't ALWAYS correct for properly is a total transit switch failure somewhere along the network connection, and data corruption inside a switch, most likely due to partial memory failures.

For the above stated reasons, end-point switches may want to employ some form of a Transport Layer Protocol machine on connections where absolute data integrity is necessary. At the 1985 ARRL Computer Network, I proposed the Amateur community adopt the use of the CCITT X.224 Transport Protocol network-wide, both for virtual circuit networks (AX.25 types) and datagram networks (the TCP/IP crowd). This Transport Layer actually defines five different classes of Transport Layer Protocols, with negotiation of classes allowed at the Transport Layer connection establishment. The various classes provide different forms of end-point error detection and correction. Interested readers should refer to the 1985 ARRL Computer Networking Conference proceedings.

The important part of the X.224 Transport Layer Protocol is that with the various classes defined, a switch can request only the amount of overhead necessary, without having to live with a lot of excess baggage. The datagram-based network will need to use the Class 4 protocol, which has all the bells and whistles. If a Transport Layer Protocol machine is deemed necessary in a virtual circuit network, Class 1 should be sufficient, especially if the checksum option is implemented.

The X.224 Class 1 machine does very little, as far as Transport Protocols go. It starts by establishing an end-point logical connection between the two end-pont devices (normally the two end-point switches). It then relies on the use of state variables and timers to detect and recover from errors, just like Link and virtual circuit Network Layer protocols. The difference is these state variables are maintained end-to-end ONLY, they are not affected by individual Link or Network Layer connections. If an intermediate switch in a network connection fails, the Transport Layer Protocol machine will eventually detect it, due to timer failures. The Transport Layer Protocol machine will then attempt to re-establish the network connection from the source end-point to the destination end-point, without any User Amateur intervention, unless requested. Lost and duplicates packets will be detected by their wrong sequence numbers. This corrects for all but one problem, data being mangled inside a switch.

The CCITT X.224 Transport Protocol has an option to append a checksum to all data packets. Using this option, data integrity can be assured. Since the Transport machine is end-to-end, the checksum is also end-to-end. This means the checksum needs to be calculated only at the two ends, not at every intermediate switch.

The Amateurs at each end of the network connection may not have the X.224 Transport Protocol in their PADs for a while, so the switch will have to provide the Transport Protocol machine. This is the way most commercial virtual circuit networks operate. The end user accesses the network via an X.25 connection. The network then attempts to make the requested network connection using a Transport Layer protocol for end-to-end data integrity, and X.75 or a similar Network Layer Protocol. X.75 is basically the same as X.25 except it operates in a "balanced" mode, while X.25 is more of a master/slave protocol. The X.25 user-to-end-point-switch connection maintains proper data flow from the user to the network, the X.75 connections maintain proper data flow between the switches involved in the network connection, and the Transport Layer connection makes sure that the data traversed the whole network connection properly.

Keep in mind that a network of X.25/X.75 Network Layer connections may be reliable enough for most communications, allowing the Transport Layer machine to be developed and refined while the Network Layer is "on-the-air". This will allow a study of exactly how much of a Transport Layer Protocol needs to be used to back-up the Network Layer connections.

Another requirement that may be imposed on the Transport Layer machine would be that of "gatewaying" between different Transport Layer Protocols. The best way to take care of this potential problem would be to negotiate to the proper class of Transport Protocol to be used from the outset of a network connection. If a Transport Protocol is implemented that does not allow this negotiation (such as TCP), there may need to be a Transport Layer gateway at the interface switch between the two incompatible Transport Protocols. Technically, this violates

the IS0 Reference Model, since the Transport Layer Protocol each end-point sees is not the same and therefore the information at each end-point may not be valid across the entire network connection. Still, if the Transport Layer gateway is written carefully this can be a viable alternative.

The Transport Layer Protocol maching should be written in a higher level language such as C since it may need to be ported to different types of microprocessors.

Network Layer Gateway machine

Just as there may be a need for a Transport Layer gateway, there may also be a need for a Network Layer gateway. This process would be able to transform packets from one Network Layer protocol to that of a different type, making each side of the gateway appear to be working with a device of the same type protocol. This Network Layer gateway will most likely be easier to write than the Transport Layer gateway, since the change would not necessarily have end-to-end repercussions. This task is made even easier if the Transport Protocol has been negotiated to an agreed upon class, eliminating the need for Transport gateways.

The Network Layer gateway machine could be written in a higher level language for portability.

Network Layer Addressing and Routing Issues

There has been a lot of lively discussion over the last year regarding the Network Layer Protocols of choice. In addition to that basic discussion there are two other related subjects that invoke an equally lively debate. Those two subjects are what network addressing scheme is to be used, and how routing information is to be stored and how routes are to be determined.

When I first suggested the use of the Amateur callsigns as the addresses for the Link Layer of AX.25, it seemed like a natural. The best part of using the callsign is that they have been pre-assigned by the FCC., totally eliminating the need for some organization to assign addresses, which was the case for the older SDLC protocols used. I believe this is still the case, even for the upper-layer protocols. However, the Amateur callsign alone may not present enough information to "help" a network connection along.

Several additional addressing schemes have been devised by the Amateur community over the last year. Fortunately, most of these do not require an organization or groups of organizations to assign network addresses. Some of the more common addressing schemes are:

A. Callsign Only.
B. Callsign Plus Area Code/Phone Number.
C. Callsign Plus Airport Designators.
D. Callsign Plus Zip-Codes (Zip Plus 4?).
E. Callsign Plus Latitude and Longitude.
F. Callsign Plus Gridsquares.
G. Assigned Numbers by a heiarchical group of organizations.

Some of the people in the commercial network world warn against using addressing schemes that include or imply routing information. I feel that while we are building the Amateur Packet Network, and until our switches are sophisticated enough to contain all the routing information necessary to route packets without any other help, we may need some routing information included in the network addresses. This is why I suggested the use of callsigns plus gridsquares in my AX.25 Level Three proposal in the Third ARRL Computer Networking Proceedings. The suggested method there was to put the Amateur callsigns in the normal AX.25 address fields, and add the gridsquare information as 0ptional facilities. This way, the callsigns will always be there, but the additional gridsquare information can be dropped when it is no longer needed.

Meanwhile, the first actual implementation of AX.25 Level Three has recently come out, and it supports callsigns plus area codes and phone numbers. Both of these systems do implicitly carry routing information, since they both carry a recognizable method of identifying where the destination station is physically located. It is beyond this paper to indicate which is better, however both are addressed in additional papers presented elsewhere in these proceedings.

The point I want to emphasize is that one reason why AX.25 Level 2 has been so successful is that the addressing issue was resolved without creating a bureaucracy to maintain records of who was assigned what address. I feel this is equally important at the Network Layer.

The routing issue is another area of great debate. Not only is the actual route determination an issue, but how the routing information is stored is also an issue.

Eric Scace, K3NA has been working on a routing algorithm that is self-generating. Whenever a packet switch comes on the air, it notifies other packet switches it hears that it is now available. All packet switches it notifies then add it to their routing database, modifying any routes that can now be run through it. All the switches involved then pass the new routes they have come up with to each other. The last step is to erase any duplicate routes in the database. This process can take a while, and since the whole routes are kept in the database, it can conceivably grow rather large. A method of shrinking this database size is to tokenize the various paths, and then expand the tokens whenever a route is looked up. The problem with most methods of shrinking the Routing database is that the routes must be expanded back whenever they are accessed, either for look-up or for database modification. This can add time to the look-up, which may be a worse situation than the large table size.

Routing is one subject that will need more work. For now, we may be better off letting the User specify the route (explicit routing). As the Amateur Network grows, a better feel for the automatic routing algorithms will emerge.

Routing algorithms could be written in a higher level language, especially ones to be used in virtual circuit switches, since they are accessed only once (during network connection setup).

Network Layer Protocol Machine

Since I am one of the prime pushers of virtual circuits, it should come as no surprise that AX.25 Level Three is my choice for the Network Layer Protocol. I feel that since virtual circuits tend to catch and correct errors when they occur, less overall network facilities are wasted correcting these errors.

The Network Layer Protocol machine should be capable of accepting multiple network connections, from a variety of other switches. Network connections that end at the switch should be passed on to the proper higher layer protocol machine. First, if a Transport Layer Protocol machine is involved, the data will be passed to it. If the switch itself is the destination end-point, the data will then be passed to the Session Layer Protocol machine for further processing. If the destination station is a Link Layer only TNC, the data must go to the Network Access PAD program for Layer 3 processing before being transferred to the Link Only TNC. If the destination station has an AX.25 Network Layer connection to the switch, the data will be passed directly to the destination station PAD.

The Network Layer Protocol machine could be written in a higher level language. The language of choice for the switch seems to be C. I see no time constraints of the Protocol requiring that the Network machine be written in assembler. The Link Layer affords enough of a time buffer.

A Network Layer management routine should be employed as an interface between the network machine and the rest of the packet switch. This management routine will monitor the Network Layer Protocol machine operation, and make minor adjustments to certain variables. Recoverable errors may also be reported to the Network Layer Protocol machine management in order to keep a record of malfunctions.

Link Layer Protocol Machine

The Link Layer Protocol machine uses AX.25 Level 2 as the Link protocol, both between the switch and the individual Amateurs, and between the switches, running under the Network Layer connections. Since there will be many devices trying to connect to the switch, the Link Layer Protocol machine must be capable of multiple Link connections, possibly coming from more than one Physical channel.

The Link Layer Protocol machine will send its received data (after it processes it) either to the Network Layer Protocol machine, or to the Network Access PAD if the source of the data is a Link Layer only TNC.

Even though packet activity operates over half-duplex channels right now, the Link Layer Protocol machine should be able to operate full-duplex. This will allow easier testing of the code now, and the addition of full-duplex channels running at faster speeds, which may be available in the not-too-distant future.

The Link Layer Protocol machine should have a separate management section that is capable of monitoring Link Protocol machine operation. This management section should be able to report the status of the Link machine to upper layers, fill requests from the upper layers for additional Link connections, and control certain Link variables to optimize Link Layer operation. In addition, these same variables should be adjustable from the Maintenance Application routines.

Even though the packet switch will essentially replace the digipeaters, there may be a need to keep the digipeater code in the switch. One case is mentioned above, that of operation between two Link-only TNC devices. The cost is nearly nothing (since digipeating is a very simple function) to keep the function active.

The Link Layer Protocol machine could be written in a higher level language, such as C, or it can be written in assembler. I personally feel that since the Link Layer is speed dependent, it should eventually be written in assembler. This way processing time in the Link Protocol machine will not seriously affect the Link Layer Operation, particularly in the areas of time-outs. This will become especially true as the speed of the Physical channels increases.

Physical Layer Support Software

The Physical Layer support software is the packet switches interface to the "real world". It is what supports the sending and receiving of packets over the Physical channels. Since this software directly supports the hardware of a Physical HDLC channel, I feel it should be written in assembler for speed. The software has to be written specifically for the hardware device anyway, so the assembler requirement is not out of line. The end result is that whole frames should be passed between the Physical Layer support software and the Link Layer Protocol machine. These frames should also indicate which channel they were received on.

While the Physical Layer support software will need to be tailored to the type of HDLC channel used, it should be able to run full-duplex for debugging, even for half-duplex RF channels.

The Physical Layer management should be able to control certain variables such as timeouts and channel speeds. In addition, access to the same variables should be available through the Maintenance Application.

Conclusions

AMRAD will be working on the next generation of packet switch software based largely on the ideas presented in this paper. We plan to have the switch code running first on a PC compatible computer, with smaller versions of the code running on the Xerox 820 and TAPR NNC systems. We feel that designing the overall switch software first, then dividing the project up fo actual coding may take slightly longer to start, but the resulting system will more than make up for the initial lag. It should work more efficiently, and should be easier to understand and interface to.

I am looking forward to reporting next year on the status of this ongoing project.

References

Fox, T., "User and Switch Packet Digital Hardware," Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986.

O'Dell, M., "An Introduction to the HUB Operating System", Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986.

Newland, P., "A Few Thoughts on User Verification Within a Party-Line Network", Fourth ARRL Amateur Radio Computer Networking Conference, ARRL, 1985

Feinstein, H., "Authentication of the Packet Radio Switch Control Link". Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986

Borden, D., "The Network User Interface", Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986

Fox, T., "CCITT X.224 Transport Layer Protocol Basic Description", Fourth ARRL Amateur Radio Computer Networking Conference, ARRL, 1985

Fox, T., "AX.25 Network Sublayer Protocol Description", Third ARRL Amateur Radio Computer Networking Conference, ARRL, 1984

Fox, T., "Annex A Through F For AX.25 Level 3 Protocol", Third ARRL Amateur Radio Computer Networking Conference, ARRL, 1984

Fox, T., "Amateur Routing and Addressing", Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986

Terry Fox, WB4JFI
President, AMRAD
1819 Anderson Rd.
Falls Church, VA 22043

ABSTRACT

There has been a lot of activity in the area of digital hardware for Amateur packet radio in the last year. I see this trend continuing over the next few years. This paper describes some of the present packet digital hardware, and gives some of my thoughts on what we will be using in the future, both for the "TNC" and the network devices.

Terminal-Node-Controllers and Packet Assembler/Disassemblers

Let me first clarify what I mean by a Terminal-Node-Controller (hereafter called a TNC), and a Packet Assembler/Disassembler (called a PAD). I have altered the meaning of the TNC from its "traditional" meaning of any device an Amateur uses to gain access to the packet network. I prefer to use the term TNC from now on to describe any device that is used to provide a level 2 only connection for some non-packet device, such as Figure 1 illustrates. Now I hear some of you saying "WRONG, WRONG, WRONG!" I know this is not the "traditional" Amateur meaning for this device, but with true networking devices on the way, I feel we need to alter the meaning of TNC to level 2 only devices.

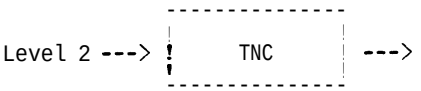


Figure 1. TNC Type Device

The reason description to level 2 only devices is that the term "Packet Assembler/Disassembler" or PAD is normally used to describe a device that makes and tears apart "packets", or Level 3 pieces of information for some non-packet device. I suggest that we adopt this PAD description for any device that interfaces a User Device to the Amateur Level 2 indicates.

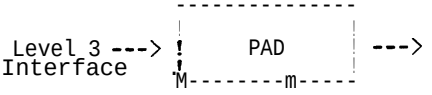


Figure 2. PAD Type Device

This separation of definitions of the terms TNC and PAD will pay off in the near future when we have both Level 2 and Level 3 devices trying to hook to the Amateur network. Since most TNCs are Level 2 only devices at the moment anyway, this should be a fairly smooth change. It should be noted that a piece of hardware could actually be one or the other of these. The TAPR TNC2 is a good example of this. Using the standard EPROMs supplied by TAPR, the TNC2 functions as a TNC. If those EPROMs are replaced by ones with AX.25 Level 3 code in them, the same TNC2 would then become a PAD for the user.

TNC's and PAD's fall into two basic categories; the dedicated box, and the computer adaptor. Until about a year ago, only the dedicated box type was available for general use. The dedicated box TNC is a separate microcomputer whose responsibility is to provide a standard terminal or computer (usually via a RS-232 serial port) access to the Amateur Packet Network. In order to do this, the dedicated box has its own CPU, RAM, program in EPROM, and a couple of serial ports (one for the packet channel, the other for the User Device interface). The advantage of this type of device is that it will run with almost any terminal or computer, and will not tie up the processing power of a host computer it is attached to.

The computer adaptor is a much simpler device, at least in hardware. It's kind of that since there's less hardware involved, more of these interfaces haven't come along. The adaptor hardware is usually nothing more than some sort of serial chip capable of supporting drivers. The problem with the computer adaptor is that the support software for Level 2 must be written for, and must run on the host processor. This will take away processing power from the host processor. Also, if the packet operation is not the end function but rather a communications medium for another program (such as a bulletin board), a major conflict can arise between the computer adaptor software and the other program.

Today's Dedicated TNC Devices

It is interesting to note that a couple of years ago, when I designed the AMRAD PAD (which was still-born) I kept saying that the number one problem with the TNCs then available was the lack of memory, both RAM and EPROM. Boy, did I get arguments about that. Well, guess what? Todays generation of TNC boards have more memory, sometimes much more.

One example of this is the new TNC+ available from Vancouver. It has eight byte-wide sockets for memory, allowing up to 64k of memory to be used, in almost any mixture of RAM and EPROM chips. In contrast, the original Vancouver TNC (the one that started it all) had only 4k each of RAM and EPROM.

Some of the dedicated TNCs available available at the time this is being written are listed below. Note that this may not be a complete list, but it will give the reader a good indication of the types available these days.

Vancouver Amateur Digital Communications Group (VADCG) has a new version of the original TNC. The TNC+ is based on the same 8085 CPU and 8273 HDLC chip design as the original. The major improvements are the use of eight byte-wide memory sockets for up to 64k of combined EPROM/RAM, and the use of a single power supply input. It is available from Lockhart Technology at the following address:

Lockhart Technology
9531 Odlin Road
Richmond, B.C. Canada V6X 1E1

Bill Ashby and Son sells a version of the original Vancouver TNC which uses a different kind of memory devices but is otherwise compatible with the original Vancouver TNC from:

Bill Ashby and Son
Box 332
Pluckemin, NJ 07978

Heathkit still sells their version of the TAPR TNC-1, called the HD-4040. This is a 6809/WD1933 based TNC. For more information, write to:

Heath Electronics
Benton Harbor, MI 49022

Another TNC based loosely on the TAPR TNC-1 design is the Kantronics Communicator. It is a 6803 based TNC, which uses a software UART instead of the WD1933 that is in the TNC-1. Apparently, Kantronics has just introduced a new Communicator, the KPC-2. At this time, I haven't seen one of these yet, so I can't tell how different it is. For more information, contact Kantronics at:

Kantronics
1202 E. 23rd Street
Lawrence, KS 66046

Another TNC that uses a software HDLC UART is the GLB PK1-L. It is Z80 based, and the low-power version draws only 25ma at 12VDC. Contact GLB at:

GLB Electronics
151 Commerce Pkwy
Buffalo, NY 14224

I should mention that TAPR is no longer selling the TNC-2 directly. They have licensed the product to several companies to produce. This TNC uses a Z80 CPU, an SIO for both serial ports, a 32 kilobyte EPROM, 8k of RAM, and the standard TAPR design Exar chip modem.

One of the companies selling the TNC-2 clone is GLB. It is called the TNC-2A.

Another company that is producing the TAPR Clone is the MFJ Enterprises MFJ-1270. Contact:

MFJ Enterprises, Inc.
Box 494
Mississippi State, MS 39762

Yet another TNC-2 clone is the PK-80, from AEA. AEA was one of the companies that sold a version of the original TAPR TNC-1, so they have been into packet radio for a while. For more info, contact:

Advanced Electronics Applications
PO Box C-2160
Lynnwood, WA 98036

Another TNC-2 clone is the TNC-200 from Pat-Comm. It is available in various forms, from bare boards to assembled units. Write to:

Pat-Comm Packet Radio Systems, Inc
4040 W. Kennedy Blvd
Tampa, FL 33609

One company, Packeterm, sells an integrated packet terminal, called the IPT. This is a whole computer terminal with a packet board included. For more info, write to:

Packeterm
PO Box 835
Amherst, NH 03031

More and more of these dedicated TNC devices should become available as packet radio continues to grow. One of the ideas I have is to design a dedicated TNC using a single chip microcontroller. Now that AX.25 Level 2 Version 2 is becoming fairly stable, I think it is time to have the protocol available on silicon. This way, anyone who wants to run the protocol can obtain one of these single-chip controllers with the code in it, and be confident that it will be compatible with others (once the initial bugs are worked out). This same device could be used stand-alone as a TNC, or as a front-end packet processor for a larger device such as a packet switch. I will be working more on this in the near future, probably based on the Intel line of microcontrollers, such as the 80C152.

The 806152 is similar to the popular Intel 8051 type microcontroller chip. It has a CPU, 8k of either EPROM or mask-programmable ROM, 256 bytes of RAM, two timers, async serial channel, and an HDLC serial channel supported by two DMA channels. This chip should allow the whole TNC to be done with about six chips, including modem.

Packet Adaptors Available

The packet adaptor devices are relatively new to the packet scene. Probably the first of these was the state machine add-on to the Xerox 820 designed by Jon Bloom, et all. Then came the FADCA/TAPR Frame Assembler/Disassembler (FAD), also for the Xerox 820. Phil Karn, KA9Q has written AX.25 Level 2 code in C for the Xerox 820. The combination of one of these two hardware mod's and Phil's software makes a good alternative to a dedicated TNC.

The first commercial packet adaptor that I am aware of is the AEA Pakratt PK-64. This device plugs into the expansion socket of a Commodore 64, and supports not only packet operation, but also RTTY, ASCII, and AMTOR. Once again, it provides a HDLC capable serial port (Zilog 8530 SCC) and modem for hardware, and canned software in EPROMs. Since the Pakratt is used only on the C-64, it can have machine-specific code in it, such as a nice menu-driven user interface.

Another packet adaptor recently announced, is one that is designed for the IBM PC and clones. It was developed in Canada, and is based on the Intel 8273 HDLC controller chip, and the Exar chip set modem. At this time, I don't have the address of where to order this device.

I have also come up with a design for a packet adaptor for the IBM PC. It uses an 8530 SCC for the HDLC channel(s). It also has room for two AMD 7910 world-modem chips, and RS-232 drivers for both channels. It is called the PC-PAD-1, and will be available from TAPR shortly.

These adaptor devices can make the entry to packet radio much cheaper if one is available for a computer you already own. As mentioned earlier, the problem with these devices is that the host computer will probably be so busy running the packet software necessary, that it won't be able to run other programs. While this is fine for an Amateur wishing to communicate over packet radio, it is not so good if the host computer must also run some other program, such as a bulletin board, or possibly some emergency database program.

Packet Switch Hardware Designs

It is likely that before long true packet switches will be replacing our present digipeaters. These packet switches will be more sophisticated devices, requiring more memory, larger program storage, and larger data storage, and more RF channels and control circuitry. For more information on what I see is needed in software, see a companion paper titled "Packet Switch Software Design Considerations" elsewhere in these proceedings.

Packet Switch Hardware Requirements

In order to properly design any complicated device, one should first look at what is ideally expected from that device. If all the functions are not necessary immediately, a graduated design may be in order. This is what I think we need to do with the design of the packet switch.

Some of the requirements for the ultimate packet switch are as follows:

- Lots and lots of RAM memory for data.
- Multiple Channel Operation (probably at least four channels, each potentially full-duplex).
- Mass storage device(s) for routing and User directory storage, plus program loading.
- Fast main processor capable of accessing large amount of memory.
- Program storage either in EPROM or mass-storage.
- All input and output should be under DMA control to reduce main processor load.
- Service processor and control system for diagnostics and control of the switch.
- Time-of-day clock, and various timed interrupts, as necessary.

-------- continued next page ---------

Local console interface for local access to switch.

Low power and/or battery back-up operation.

Lots and lots of RAM

The amount of RAM depends on how many stations are going to be using the switch, how the switch software is written, and to some degree what protocols are to be used. A full level 2 AX.25 frame uses about 275 bytes without digipeaters, or up to 335 bytes with the full complement of digipeater addresses. If one allows for a window of 7 packets in each direction (transmit and receive), this can mean from 3.6k to 4.7k bytes per user, just for actual data buffers. There will also be some small amount of storage per user at each level of protocols used. With between 4 and 5 kilobytes needed per user, one can see that the more the RAM, the better in a packet switch. Obviously, this is a worst-case scenario, and not all this memory must be reserved per user at all times. A guess on my part is that about 20 percent of the absolute amount of memory would actually be needed. If this holds true, 128k of RAM could actually support up to 150 users rather than the 32 users suggested by setting aside 100 percent of memory for all users.

Of course, this evaluation does not hold true if a lot of data latency is encountered inside the switch. As an example, say a switch receives a lot of packets from a 56 kbps channel, and finds that the majority of the data is to sent out an HF channel operating at 300 bps. This means much of the data must be buffered for a longer time while the level 2 protocol tries to shove the data over the HF link. If the switch starts to run out of memory, it may have to begin to flow-control data coming into it until the HF link frees up some buffers. This may cause back-pressure of data along the Network as more and more switches have to hold data until the next switch can accept it.

Another use for memory in switches will be to hold the operational routing tables, which can grow to rather large sizes. The reason for maintaining the operational routing tables in memory rather than a mass-storage device is due to the need for continual, fast access to them. A back-up version of the tables should be maintained on a mass-storage device (for power-failure or data corruption protection), but the main, active tables should be in RAM. This adds another, rather significant memory requirement to the switch design.

These days memory is fairly inexpensive, especially dynamic memory. It therefore makes good sense to me that a switch have as much RAM as possible, preferably as much as the CPU can easily handle.

Multiple Channel Capability

Before long, packet switches may have to monitor several different RF channels, much as the dual-port digipeater does now. Some of these channels may be operating full-duplex, especially the higher speed ones. As a guess, I think four to five channel capability should be enough for the most part, with the possibility of expanding up to six channels. The higher speed channels should either use DMA directly, or use a smaller TNC-type device to pre-process the level 2 connections, thereby relieving the main processor of some processing.

Mass-Storage Device(s)

Once the Amateur packet network begins to expand, there will be a need for some sort of mass-storage of data at each switch. One of the things needed to be stored at each switch is a routing table that the switch can access to decide how to route packets. Another thing the switch may want to store is a file containing the list of users it regularly services. Also, the switch may keep a couple versions of its operating program in case it need to re-boot for some reason.

FastCentral Processor

In order to build a switch that can process data quickly, a fast central processor (CPU) should be used. Since newer, faster CPUs are constantly being developed (which are usually very expensive at first), we may not be able to afford the "best-of-the-line". An important factor when choosing a CPU is that the development tools must be readily available for designing both hardware and software around the chosen CPU. Some of these tools include simple hardware interfacing to standard components, and Assembler and C compilers for software, at a minimum. The best CPU in the world isn't much good if one must write all code for it in hand-assembler.

Program Storage in EPROM or Mass-Storage

There should be room in the switch for a minimal amount of bootstrap code in EPROM to load the main operating program either from a mass-storage device, or possibly from a neighbor switch. Another alternative would be to have the main operating program run directly from EPROM, but this would make remote updating of switch software more difficult. since switches may be at remote locations, some method of remote loading of switch code may come in handy.

All I/O Should use DMA If Possible

As mentioned earlier, all input and output devices should operate using Direct-Memory-Access (DMA) if possible. This would include packet channels and any mass-storage devices used. The use of DMA relieves the main CPU of having to handle constant asynchronous data interrupts, or worse, polling of I/O devices, both of which can waste a lot of CPU time. DMA chips are available today at an inexpensive enough price to make a hardware design using them cost-effective.

Service Processor and Control System

When switches get more complicated, it may become necessary to provide some method of monitoring switch operation and controlling the switch if a failure is detected. One way to do this would be to provide a second, smaller micro-computer system to monitor the packet channel(s) and/or the main switch processor bus for normal operation. This separate system would have its own distinct identity in the network, and should be designed so that the main switch could not block operation of the Service Processor. If a failure is detected, the Service Processor could have several options programmed into it, from simply re-booting the main switch to taking the switch completely off the air and notifying someone of its actions and why.

This type of device is commonly used on larger computer systems to monitor the main computer, and I think we should include some version of this device in our more elaborate switches. Since the main switch and the Service Processor are inter-related, they would have to be designed with each other. At the moment this device is a luxury, but it should be kept in mind for future switch designs.

Time-Of-Day Clock and Timed Interrupts

In order for a switch to accurately control the various protocol operations, some sort of timer is necessary. Most computers use one or more timed-interrupts to control these functions. In a multiple-task computer, times cannot be regulated by software timing loops, so these timed interrupts are important. For example, the TAPR TNC2 uses a 600 Hz interrupt, while the Xerox 820 computer board uses a once-a-second interrupt. These are generally considered the two ends of the spectrum for timed interrupts, with the TNC2 being almost too fast, and the Xerox 820 being too slow.

The time-of-day clock is less of a necessity, but could come in very handy, especially if satellite or HF access is to be provided at the switch. It also may be handy for diagnostic purposes to time-stamp certain operations of the switch, such as routing table updates and last access by individual users. Once again, the cost of adding this feature is very small, so why not include it? Most time-of-day clock chips available these days also provide timed-interrupts, so both of these can be had for the price of one chip.

Local Console Interface

It is inevitable that switches are going to break down eventually. In order to work on them, or just periodically check on their operation, a local console interface should be provided. This local interface should be at a very base level of the switch (directly into the CPU if possible) to reduce the amount of potential problem points between the console and the local console while debugging.

Battery Back-Up Operation (optional).

There is a rapidly growing interest in the use of packet radio by emergency groups. In order to help provide emergency communications, the packet network should be capable of operating for some period of time either on batteries, or through the use of generator power. While this isn't a definite requirement, if the switch is designed with battery operation in mind (such as the capability of running off of 12 volts DC), it will be much easier to add later.

The above items do not constitute a complete list of design goals for a switch. The radios, for example, are partially covered in a separate paper, found elsewhere in these proceedings.

With the above goals in mind, I will now turn to how I see the packet switch design evolving over the next few generations of hardware.

Packet Switch Based On TAPR TNC2

The first step of the packet switch Thanks to Howie, N2WX, several of us in AMRAD (as well as others) have been playing with AX.25/AX.75 networking on a local basis for several months now. The packet switch code actually resides inside a TAPR TNC2. I must point out that this is by no means a fully operational internetwork switch capable of auto-routing and lots of other fancy stuff, but it IS an actual Network switch, not a digipeater. Full AX.25 level 2 connections are made with each user, and the switch does move data from user to user based on AX.25/AX.75 level 3 connections. As a Local Network Controller for a limited number of users it does indeed work.

The amount of RAM on a present TNC2 is 16 kilobytes, with 32k of EPROM for program memory. The TNC2 uses a Z80 CPU, and an SIO for both terminal communications and packet communications. This amount of memory (especially the RAM) is sufficient to experiment with, but could hardly support the total number of users on even a semi-busy packet channel. Since there is no mass-storage device interface on the TNC2, some of this RAM would have to be used up to hold the routing table for inter-switch connections, reducing even further an already scarce commodity.

Also, since there is only one (or with modification and loss of the terminal interface, two) packet interface(s), a switch based on the TNC2 could only operate directly on one packet channel, without getting into clever data switching arrangements which would be a kludge. The maximum speed of the TNC2 serial channels is probably around 20 kbps, allowing it to be used on our medium-speed channels.

The packet channel interface operates in an interrupt mode, rather than using DMA. This is not really a problem though, since there is only one packet channel, one terminal channel, and no mass-storage devices to worry about. As mentioned earlier, the TNC2 has timed interrupts at a 600 Hz rate, relieving the software of a lot of timing overhead.

The TNC2 uses a 12 Volt DC nominal power supply, drawing from 150 to 300 ma, which makes it very good in the power department.

Packet Siwtch Using the Xerox 820

Another alternative for a packet switch from the 8-bit world involves the use of the Xerox 820 boards. These boards were surplused by Xerox after the 820 was discontinued, and a large number of them found their way into the hands of packet experimenters throughout the country. Some of the advancements in packet radio operation in the last year or so (such as the WORLI packet bulletin-board software and the dual-port digipeater) are directly related to the availability of these boards. It has on it a Z80 processor, two 2716 EPROMS, 64k of dynamic RAM, two serial channels, a parallel channel, four timers, a floppy-disk interface, a parallel keyboard interface, and a video interface that emulates an ADM-3 terminal. Many of these 820 boards are already out there as either PBBS systems, or dual-port digipeaters linking two different packet operations together.

It is natural that the Xerox 820 be considered as a possibility for the basis of a packet switch. In fact, at the September 1984 meeting of the ARRL Digital Committee, one of the requirements made for the evaluation of Virtual-Circuit network design versus Datagram network design was that they both be made to run on a Xerox 820 board.

The Xerox board's 64k memory space is better than the TNC2's 16k. One drawback to this is that a Z80 processor can only access a total of 64k of memory at a time, including both program and data space. This means that the more memory a program takes, the less that is available for data. This implies that the program used should be as small as possible to save more memory for data.

The Xerox 820 includes four timers, two of which are used to generate one-second interrupts for the 820's monitor. The other two timers are available to the user. One of these will probably be needed as a prescaler for the other in order reduce the number of interrupts per second achieved. The once-a-second interrupt could be used to build a software time-of-day clock, although it would be susceptible to inaccuracies due to periods when interrupts are disabled.

The serial ports on the Xerox 820 are interrupt driven, using an SIO chip. TAPR has an add-on daughter board available that removes the parallel interface and replaces it with two more serial channels, using an 8530 SCC in interrupt mode. This means that one Xerox 820 can support up to four different packet channels, which might come in handy at a central packet switch location.

While the 820 board does have a floppy-disk interface, it is only capable of handling single density operations. This means a total of 241 kbytes of storage on an eight-inch drive, or about 160 kbytes on a double-sided five-inch floppy. There are several daughter board systems commercially available that replace the single-density controller chip on the 820 with a double-

capacity to 390 kbytes of storage. Some of these

incompatibility has been noticed, particularly in These incompatibilities can be overcome without too much difficulty, and the added storage is a welcome addition.

When using CP/M on the Xerox 820, there is a wealth of software tools available for switch software development. All kinds of assemblers and higher language compilers (such as C, Pascal, Fortran, and yes, even BASIC) are available. In addition, there are a large number of debugging systems available to the programmer, so the 820 can be considered a good choice to develop code on.

I still think the Xerox 820 is a good choice for smaller switches, where there will not be a lot of users. The boards usually came built, with most of the ICs soldered in place, so the number of problems occuring with a switch based on an 820 will probably be less than some other designs. (It is interesting to note that half the problems we have experienced with the WB4JFI-5 digipeater are due to corrosion of IC socket-to IC pin connections, and that as soon as the ICs are re-seated, the problems disappeared).

Packet Switch Design Based On the TAPR NNC

A short time ago, TAPR came up with a design for what they call the Network Node

Controller, or NNC. This is the same thing as a packet switch. The NNC design is based on a Z80 like chip made by Hitachi (the HD64180). This CPU executes the Z80 instruction set, plus a few new instrucions that have been added. In addition, it includes two channels of DMA, 2 sixteen bit timers, two async serial channels, a built-in clock generator, and can physically access up to 512 kbytes of memory (through a 64 kbyte logical window). It also has a built-in interrupt controller, and dynamic memory refresh controller.

TAPR's design presently consists of three boards. The first board includes the 64180 CPU, four HDLC ports (using two SIOs), a parallel printer port, a battery backed-up real-time-clock, a SCSI bus interface, and up to 512 kbyte of memory via byte-wide sockets, half of which may be battery backed-up.

The second board is a piggy-back floppy disk controller board. This board will allow the connection of high-density disk drives for those wishing to do software development directly on the NNC, via CP/M or similar operating systems. It will also come in handy to hold those routing tables.

The third board contains four modems of the Exar 2206/2211 variety. TAPR sees these being used as an HF modem (300 bps, 200 Hz shift) and a VHF modem (typical 1200 bps, 202 type), with two additional modems for future use. To me, these modems are still the weak link of TAPR designs. The PLL modems are inferior in operation to standard filter-type modems, and require periodic re-alignment (I just had to re-align my first TNC2). I am not sure how these modems would survive in an uncontrolled enviroment, such as where a digipeater or packet switch might be located (WB&JPI-5 operates in a metal shack part-way up a broadcast tower and the tempature inside the shack goes from about 115 degrees in the summer to below zero in the winter).

The NNC has much of what it takes to build a decent packet switch. The main limitation is still the RAM memory access. Since the CPU is based on a Z80, it can only access a total of 64k of memory at a time, both program and data. Even though the 512 kbytes of memory can be paged through in 64k windows, the amount of program space must be included in that 64k, again reducing the amount of data space directly accessible. If the program takes 32k to run, this leaves 32k total data space available without paging. When the program grows to 48k, the data window is then reduced to only 16k chunks.

This memory constriction may not be much of a problem for quite a while, since paging of data chunks can be accomplished without too much software magic. The SCSI bus allows for larger mass-storage devices than floppy-disks, such as hard-disks. The two timers will allow for timed interrupts, and the time-of-day clock is a nice addition.

Since the NNC is software compatible with the Z80, once again the whole CP/M software family is available for program developers.

The NNC is desfgned to use low-power devices, such as static RAM and SIOs instead of SCCs, the latter of which is becoming easier to use. Because of this, the NNC is a good candidate for very-low power operation, such as mountain-top switches.

I see the TAPR NNC as being a potential stepping stone on the path to the more complicated packet switch. Since it is basically software compatible with the Xerox 820 board (except at the lowest port address level), code developed for the 820 will be able to be run on the NNC. This will allow packet switches to be upgraded fairly easily when the primary reason for obsolescence is due to the lack of memory. It will also come in handy when the supply of surplus 820 boards dries up.

Packet Switch Design Based on IBM Clones

Over the last six months there has been a fairly quick conversion of the AMRAD crowd from CP/M systems (either 820 boards or S100 systems) to the IBM world. Yes, even I, the staunch S100 supporter has yielded to the tidal wave of big blue. In addition to my three S100 systems and 0's, I have two IBM-PC compatible computers (one a real IBM, and the other a clone). In addition, I am trying to put together a transportable PC clone, similar to my transportable 820 system (the size of a Kaypro, complete with TAPR 8530 mod and 7910 modem built-in). What has caused this wave on blueness? It's a simple matter of cost. The price of IBM-PC's and the PC clones has dropped to the point where one can build a complete computer for under $800.00, including monitor and drives. With all that software out there, how can you go wrong!

The idea of using a 16 bit CPU for the packet switch is not a new one, it's just that until recently the cost was too high. Now wait, I hear you saying that the PC is not a true 16 bit system. You are right. You are also correct in saying that even though the PC can address over a megabyte of memory, it still must do it in 64 kbyte pages. There are important differences between the PC and the normal eight-bitters.

First of all, even though the 8088 CPU looks to the outside world like an eight-bit CPU, it thinks inside with a full sixteen bits. This means instructions can be executed quicker. In addition, it's instruction set has been improved, allowing some functions to take fewer instructions to execute, thereby speeding up processor throughput.

Secondly, even though the 8088 can only work in a maximum of 64k banks, it can use different 64k banks for program and data. In fact there are four independant 64k banks that can be anywhere in the one megabyte address space. One of these segments is used for program operation, another is used for data storage, a third is used for the stack, and a fourth is provided as an extra segment. Since each of these segments is completely independant of the others, adding to the computer program does not reduce the amount of data memory directly accessable. This can be an important point when dealing with larger numbers of packet users.

In order to use the PC or PC-clone on packet, either a separate TNC type device must be used, or a HDLC capable board must be added to the PC system. We have designed a PC-compatible board that plugs into the bus that has an 8530 Serial Communications Controller (SCC), which has two programmable serial channels. Each of the two channels can be strapped for either RS-232 operation or can be used with a AMD 7910 World-Modem chip. This board (called the PC-PAD-1) fits in a short-slot position, so it is usable in almost any PC-compatible computer. The PC-PAD-1 should be available shortly from TAPR.

A follow-on project to the PC-PAD-1 board is also in the works. Due to limitations of the PC (lack of slots, interrupt lines, and DMA channels), and since the PC-PAD-1 only occupies a short slot, I felt it would be a good idea to expand on the basic board. In order for a switch to perform added functions such as acting as a gateway between networks on different frequencies, it will probably need more than two HDLC channels. Therefore, the next generation board will have more SCC chips (at least three), it's own interrupt processor chip (slave to the PC's interrupt processor), and at least an additional 7910 modem. How much the board will end up having on it is based on how much we can stuff onto a standard PC card. I have already heard from a couple sources that a minimum of six separate HDLC channels would come in handy. The planned name for this board is the PC-PAD-2.

Ideally, it would be nice to have the HDLC ports interface to the PC through DMA channels, thereby reducing CPU overhead. Unfortunately, the PC only provides for a total of four half-duplex DMA channels, and this cannot be expanded without cutting/kludging motherboard. In addition, one of these four DMA channels MUST be used for dynamic memory refreshing. Two of the remaining DMA channels are reserved for mass-storage devices, if present. Since the switch should have some sort of mass-storage, at least one of these (and most likely both of them) will be used up. This leaves only one half-duplex DMA channel available on the standard PC, which is essentially worthless for

packet work. In order to turn the one half-duplex channel around, and arbritrate which device is using it, more processing overhead would be required than to simply use interrupts in the first place. The bottom line is that DMA is not real useful for packet I/O processing on the PC.

Another limitation in the PC is in the area of interrupts. The motherboard has an interrupt processor capable of arbritrating up to eight interrupts. All of these are potentially reserved in the PC for certain devices as follows:

IRQ0	System board Timer 0.
IRQ1	Keyboard Scan Hardware.
IRQ2	Optional Clock Module.
IRQ3	Secondary Serial Channel (COM2).
IRQ4	Primary Serial Channel (COM1).
IRQ5	Optional Hard-Disk Controller.
IRQ6	Floppy-Disk Controller.
IRQ7	Printer Port.

Since they are all reserved, but we still need at least one, something has got to go. What I have done with our SCC board was to use the secondary serial port interrupt first, followed by the primary serial port. The only other interrupt that could be "obtained" without leading to conflicts would be the printer port line. I wouldn't want to take up either disk interrupts, and I also think a clock module is going to be an essential part of the packet switch.

Once an interrupt line is made available from the master interrupt controller on the PC motherboard, it is possible to add one or more slave interrupt controller chips to that line. In order to provide multiple HDLC channel devices (such as more than one SCC), the use of a slave interrupt controller will help to figure out which device caused an interrupt. Rather than polling each device, the interrupt controller is polled to see which device caused the interrupt. Since the slave interrupt controller will be on a plug-in card, as many of the interrupt sources as possible should also be on that card, which will reduce or eliminate the need for inter-card wires. The use of slave interrupt controllers is not the best way to implement intyerrupts, but in the PC enviroment it is the best available, in my opinion.

The power consumption of the PC-compatible computers varies with what is plugged into the bus, and what type of mass-storage device is being used. Most of the IBM computers were sold with 65 watt switching supplies, which provide enough power until the cornputer is full of RAM, has both floppies, several additional cards, and an internal hard-disk. This tells me that a typical packet switch computer based on the PC would probably draw about 50 watts on the high-end. While this is higher than some would like for a mountain-top switch, it is still fairly low power-consumption when one considers the computing power involved. The replacement of standard TTL devices with high-power CMOS, and the removal of local console devices (such as keyboard and video board) would help to reduce this load.

Packet Switch of the Future

Down the road, it would be nice to have a packet switch designed by the Amateur community. What follows is some of my thoughts/ideas/leanings on what a computer specifically designed as a packet switch might look like. Some of this is taking shape now, while other parts are just in the initial idea stage.

One of the most important concepts I want to push from the outset of this section is that eventually a single CPU will most likely become overloaded with work in a switch, especially a switch that covers a metropolitan area, or a switch that interfaces between several different subnetworks. The method I propose to overcome this situation is to break up the workload of the switch into several, smaller pieces. Some of theses pieces would then be taken care by smaller, slave processors to the main switch processor. This way the switch can handle its functions more effectively without being interrupted to do the more mundane work. An example of this whould be a slave board that would handle one or more HDLC channels. This board would be very much like a TNC, and would be in charge of maintaining the AX.25 level 2 connections. It would pass data to/from the main switch only after it had processed all the level 2 work necessary. This relieves the switch processor from having to do ANY Level 1 or Level 2 processing.

Central Processor Used

In order to take advantage of the development tools we presently have, I decided that my design for the next generation packet switch should be based on the Intel line of microprocessors. My feeling is that it should be a true sixteen-bit processor, not an eight-bit external one. While it would be nice to use either the 80286 or 80386, both the cost of theses chips, and the added complexity of the hardware involved detracts from their use. Because of this, at this point, I think the Intel 80186 CPU is the best compromise to be used in the packet switch. Most of the C compilers and assembler packages for the PC now include 80186 support, so the development enviroment is there.

General Switch Computer Architecture

There are as many different approaches to the design of a computer as there are designers. In my opinion, one of the foremost requirements is that the switch be designed to use a motherboard with a bus to plug special function boards into as needed. This allows the switch to change as needs change, without obsoleting the hardware. It also allows for easier troubleshooting, since function boards can be swapped-out until the defective one is found. As to which bus should be used, at this point I am leaning toward a modified version of the IBM PC AT bus. It is both eight and sixteen bits wide, allowing present PC boards to be used for the more mundane operations, such as video and disk controllers. The AT bus also has twice the interrupts and DMA channels available as the PC. The reason I say a modified PC AT bus is that the AT still does not allow for expansion of the DMA channels to the function cards. The switch should provide the necessary signals on the bus to allow DMA controllers on the plug-in boards in addition to what is used on the motherboard. Since the AT bus does not have these signals, some method of supporting them would have to be added. It appears to me that this is a small price to pay for all the support that using the AT bus would bring.

The switch motherboard should also have the hooks needed to add a "Service Processor" board. The Service Processor is kind of an advanced watchdog device that should continually monitor the main switch operation for abnormalities. Since it will need direct access to certain points on the motherboard that are not on the bus, some additional connections will need to be provided.

HDLC Channel Processor Plug-In Boards

There are actually two different types of HDLC channel boards I see being used in the packet switch. The first version would be similar to the PC-PAD-2 described above. The main difference would be that the board would use a dedicated DMA controller for data transfer to the switch rather than interrupts. This would greatly relieve the central processor from byte-by-byte interrupts from each HDLC controller. I would recommend that the HDLC chip used be the 8530 SCC chip. I feel it is a superior device, and performs well with the Intel line of chips.

The second and more advanced type of HDLC controller board would be a slave processor that would use an on-board CPU such as a 280 to process the data of the HDLC channel(s) it supports, and use DMA to pass the data to the central processor of the switch. As mentioned above, this slave processor would actually handle all of the AX.25 Level Two connection overhead, further relieving the switch. This is needed as the HDLC channels go to faster and faster speeds. There comes a point where one processor will become bogged down, no matter how fast it is. This delegation of responsibilities to slave processors will become a necessity eventually, and if the switch is designed to accomodate it from the beginning, so much the better.

Both types of HDLC boards should have a watch-dog timer on every HDLC channel they provide. It would also be handy if the watch-dog timers had outputs that fed the switch processor, the HDLC Channel Processor, or a Service Processor, that indicated when they had timed-out.

Mass-Storage Devices Support

If the PC AT bus is used, standard IBM compatible mass-storage devices could be used, relieving the designer of both hardware, and more importantly software work. Both floppy-disk and hard-disk support should be available. One point is that at least the floppy-disk system should probably NOT use a high-density recording method. In order to make the floppy more reliable, lower density storage methods should be used, and if more storage is needed, either more drives should be added, or a hard-disk should be considered. The system should also be designed so that as much data is maintained in RAM memory as possible, with the mass-storage used primarily as a data back-up.

Memory Requirements

The memory requirements for a packet switch falls into three catagories; program memory, data memory, and control and variable storage memory. Each of these types of memory have different needs, so they will be discussed separately.

The program memory would be responsible for holding the program(s) that the switch executes. It may hold the actual executing code itself, or it may contain a copy of the code that is read out of the program memory into another part of the switch memory for actual execution. I anticipate what will eventually happen is that a small "bootstrap" version of the operating program will be kept permanently in the program memory in case the switch needs to be completely re-booted. The actual operating software will then be loaded into the switch, either from a mass-storage device, or over the radio from another switch or switch control operator. Since this boot code must always be available, it should be stored either in EPROM (prefered) or battery-backed RAM. Once the real operating software is loaded into the computer and is executing, this EPROM code could then be phantomed out, reducing the amount of memory space occupied.

The data memory will occupy the most address space of the switch. The simple rule regarding data memory is; The more, the merrier. Since there will be a lot of data memory, it should be fairly inexpensive. I am leaning toward using dynamic memory for this part of the switch. These days, a megabyte of memory only costs about $120.00. If dynamic memory is implemented, I would also recommend that a dynamic memory controller chip be used, rather than the DMA refresh approach that IBM used. The switch will be busy enough already, without wasting additional overhead doing memory refresh. The cost of a dynamic memory controller is not much these days, so it would be money well spent.

The dynamic memory support should also have some method of detecting memory failures. One method is to add a simple parity check scheme similar to what the PC implements. Another method is to use one of the more advanced dynamic memory controllers that include failure detection. These more advanced controllers can not only detect failures, but "hide" the bad section of memory from the host processor, something which might come in real handy for remote, mountain-top switches.

Another idea associated with the main data memory is to have some spare amount of it available. If a memory failure is detected, this spare memory could be placed into the memory map of the host processor instead of the bad memory. The size of this spare memory could vary from a spare bank of chips on the motherboard to a completely separate, full amount of data memory with it's own memory controller on a add-on board, in stand-by. Obviously the latter is an extreme, but it would be advantageous where the switch is at a real remote location with very limited access.

At this point in time, the chips to use would seem to be the 256k by 1 devices. They are cheap, and have a very high density. A recommended dynamic RAM controller is the Intel 8208, since it works well with the 80186.

The third type of memory is used to contain the important variables and control information used by the switch, such as callsigns, link connection information, switch status flags, etc. This memory should be battery backed-up, and it should have a checksum which is updated whenever the data in the memory is changed, and is periodically checked. This memory is important enough that an added measure of safety might be required, that of maintaining a second copy which is also updated and checked. This way, if there is a failure in the primary memory module, the back-up could be switched into operation. Since this memory is battery backed-up, it should use a very low amount of power, such as a CMOS device. Fortunately, the amount of this memory needed is relatively small.

Service Processor Board

The Service Processor is an optional board that should have a separate connection to the motherboard. The function this board provides is to perform sanity checks on the packet switch. Exactly how interconnected the main switch motherboard and the Service Processor board are depends on how many sanity checks one feels are needed. At a minimum, the Service Processor should receive periodic "pings" from the switch processor via a common I/O port and also monitor data and/or address lines for a "stuck" processor. In addition, the Service Processor might also monitor each HDLC channel for 'stuck" transmitters, possibly indicating that HDLC channel is not being properly serviced.

Whenever the Service Processor detects an error condition, it should be capable of taking one of three actions; if the error is easily recoverable the Service Processor should attempt to correct the problem, if the Service Processor cannot easily correct the situation it should perform a hard reset to re-boot the switch, if the Service Processor determines that the problem is a major one, it should disable the switch from transmitting on any of its channels. In addition, the Service Processor should store the symptoms of the failure for later retrieval during servicing. Also, the Service Processor should have a distinct packet address, capable of sending and receiving connection requests. If the Service Processor detects a switch failure, it should notify neighbor switches or a service point.

In order for the Service Processor to attempt to "heal" the packet switch during minor problems, it should be able to control certain parts of the packet switch. It should be able to change or disable certain parts of memory if it thinks there is a memory problem. It should also be able to cause the packet switch to cold-boot if necessary.

An obvious potential problem can be if the Service Processor itself goes bad. In order to prevent a bad Service Processor from messing up the switch, some sort of hardware interlock should be provided on all interfaces to the switch. This might include a timed window during which the Service Processor can send commands to the switch. Any commands that fall outside this window would be ignored by the hardware.

This timed-window could be designed such that in order for the Service Processor to gain access to a critical area of the switch hardware for alterations, it must first send a byte having a certain bit pattern to one of it's own ports. Hardware at the output of that port compares the received byte to what it expects, and if they match, unlocks access to the requested switch area. This access is only allowed for a short period of time (using a hardware timer), and if the Service Processor fails to send it's command before the time ends, the door is closed, and the Service Processor's request is ignored.

Additional Slave Processor Boards

There might be the need for additional slave processor boards in the switch to perform

functions such as User directories, and more importantly routing database upkeep. Once again, this might best be handled by having a separate slave processor, such as a Z80, with a lot of RAM and a canned database program. This way, when the switch needs to obtain the necessary routing information, it can ask the slave Route Processor for the route, then continue with other tasks while the Route Processor looks up the route. Another advantage is that the routing information is stored in RAM for faster access, but still won't take up valuable switch memory space.

Power Consumption of the Advanced Switch

Even a quick look at the above will indicate that our packet switch has become a power hog. In order to reduce the amount of power used, the switch should be designed using CMOS devices wherever possible. There should also be some way of shutting down selected portions of the switch when they are not needed (such as the local console). Even so, the switch will draw more power than some would prefer in certain situations. For this reason the programs written for this switch should allow for smaller versions of the switch, with the software testing to see if certain parts of the switch are present.

An example of this would be if a switch to be placed on a mountain top will only have a couple HDLC channels coming to it. Since the switch is powered by batteries (trickle-charged by solar cells), it is decided that the main processor can perform all its duties and also support the HDLC channel processing required. This means the HDLC Channel Processor will be removed, and a simple HDLC Controller card will be put in its place. Now, if the switch goes down and gets re-loaded from a neighbor switch, the newly loaded code must check to see what hardware configuration is being used before it starts bopping out packets. This is one example of how hardware "pruning" might affect software operation. Anytime the hardware is reduced, one should look for situations to develop in other areas. Sometimes it might be better just to increase the amount of batteries and solar cells and live with the added drain.

Advanced Switch Hardware Conclusions

While it is beyond the scope of this paper to set down the absolute design of the next generation of Packet Switch hardware (especially since we don't have much of a feel for what is actually needed yet), I think the above ideas will aid in the design of the future packet switch. One point I must emphasize is that the switch can be designed using the most sophisticated processor chip in the world, but if the development tools (both hardware and software) aren't there to support the device, no one will use it. This is the basic reason I decided to push the 80186 at this time. If other computers based on other CPU's such as the 68000 become a dime-a-dozen like the IBM PC and it's clones have, then I would be in favor of using those CPU's. For now, with the PC being a clear winner in the popularity contest, I am keeping to it's family of chips.

Another important point is that the switch be based on a bus structure. This will allow easier switch expansion and troubleshooting.

The third important point I want to leave the reader with is that even though right now it might be difficult to imagine how a single processor might get bogged down in a switch, that time will come all too soon. If the switch is designed from the beginning to off-load as much specialized processing as possible (ala HDLC Channel Processors and Route Processors), it will take a lot more to overload the switch with work. This will become a very important point in the next few years.

I hope to be working on the more advanced packet switch design over the next year. So far, my design is based on the 80186 CPU, a megabyte of dynamic RAM supported by an Intel 8208 dynamic memory controller, some EPROM, ten expansion slots, a timer chip, expandable DMA capability, and expandable interrupts. The system is NOT up and running at this time, but I hope it will be shortly. The first add-on board will be a DMA-supported HDLC board, with multiple SCC chips.

Conclusion

Developments in the area of digital hardware for Amateur Packet Radio are progressing smoothly. New designs are coming along, for the end-users, digipeaters, and the packet switches. Hopefully, in the next year the software and RF hardware parts of the hobby will catch up with the digital hardware strides made in packet radio.

References

Fox, T., "Packet Switch Software Design Considerations", Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986

Fox, T., "RF, RF, Where is My High Speed RF?", Fifth ARRL Amateur Radio Computer Networking Conference, ARRL, 1986

AN INTRODUCTION TO THE HUB OPERATING SYSTEM

Mike O'Dell, KB4RGM
13110 Memory Lane
Fairfax, VA 22033

Introduction

The AMRAD Packet Switch marks a significant departure in the evolution of Amateur packet radio. Historically the computers used for packet service (TNC's and 'digipeaters) have been small, memory-poor, and worked at their limits to do the job. But they were inexpensive, and "cheapness" was an absolute requirement if packet radio were to ever be successful. The demands of creating a transit backbone for connecting local area networks far exceed the capabilities of the first-generation 8-bit machines, so while cheapness is still very important, the AMRAD switch is being designed around an IBM PClone. Further., as the cost of higher bandwidth machines continues to fall, easily rehosting the switch as hardware advances is an important consideration. With the hardware base for packet communications moving forward. AMRAD started looking for a similar vehicle on which to base the software necessary for a serious, high-performance packet switch. The Hub system is the current choice for the underlying operating system. It is very fast, simple, and is written in C. It has the capabilities needed to do the job, and will easily outlive any hardware hosting it.

Not too long ago, software was relatively foreign to many amateurs. While this has changed dramatically, operating systems beyond the scale of a few utility routines burned into a ROM, are probably not widely understood. The purpose of this document is to discuss the architecture, motivation, and implementation of the Hub system chosen for the Switch. To accomplish this without addressing only those amateurs already versed in operating systems design (wizards), a brief tutorial overview of "traditional" process-based operating systems design theory is provided. It cannot be complete, but a bibliography is provided for those interested in entering this wonderful, subtle world. After this introduction to operating systems, we will discuss the Hub in more detail. While there may be sections whose detail is necessary but inscrutable to operating systems newcomers, such readers are encouraged to skim for the highlights and press on, reading for the general philosophy. But before jumping into the technology, we provide a short historical recitation, and roll the credits.

History and Credits

The basic architecture of the Hub was created circa 1975 by Gary Grossman at the Center for Advanced Computation (CAC) at the University of Illinois in Champagne-Urbana. The people at CAC were involved with ARPAnet since "before the beginning," and were quite experienced with communications programming, protocol implementation, and "message-based" designs. (The CAC group put the first minicomputer on the ARPAnet.) While message-based designs were elegant and simple on the blackboard, they were quite inefficient in practice. On the other hand, the industrial "real-time" community built systems which could go very fast, but were nearly unmaintainable because of their intricate, monolithic design. Grossman was looking for a way to combine the efficiency of the industrial designs with the elegance of message-based schemes. The Hub was his solution. The general Hub architecture has been used over the years in several large, successful communications projects by several companies, all of which have their roots in the Center and Grossman's luminary visions. The system described here is loosely based on the system described in a thesis by [Masamoto] (Grossman's Masters student), mixed thoroughly with good ideas borrowed from other sources, and tempered with hard-won experience gained in other network projects.

An Overview of Traditional Process-based Systems

This section is intended to be a very brief excerpt from basic operating systems theory. The experienced reader can probably proceed to the next section, while the newcomer is commended to the Bibliography, particularly [Comer], [Hansen], [Organick], [Jansen], and the-like.

There are many facets of an operating system, but for the purposes at hand (communications programming), the most important functions provided are:

1 - Concurrency

This is how the operating system creates the illusion the computer is doing several things at the same time. It is not; the processor is switching between several different jobs fast enough that for most purposes, the concurrency is real. Concurrency is also known as "multiprogramming", and sometimes as "multi-tasking", although some people erroneously think these are different.

2 - Resource Management

If several activities are proceeding concurrently, it is inevitable that left to themselves, two activities will fight over some resource, be it memory, an I/O device, or access to a third activity. The resource management function coordinates the use of resources which are either scarce, or which must be shared in an orderly fashion to accomplish the required functions.

The texts cited above will spend a great deal of time discussing these issues, as there are a great many details to get straight. We intend to gloss over most of these details and go for the general philosophy.

Before explaining the traditional model of concurrency, it is valuable to briefly discuss why it is needed. At the top level of detail, any system only does one thing: that which it does! But accomplishing that "one" jobs often involves many other small jobs largely unrelated to one another, or related only at the "edges." Instead of creating one giant program which does everything, it is easier to organize a system as a collection of smaller programs which operate independently but cooperatively to accomplish the job. The execution environment of these programs must provide some mechanism for independent execution, as well as mechanisms for cooperation. This "execution environment" is called an Operating System. They come in many shapes, sizes, and flavors, but all exhibit a basic set of characteristics. The most fundamental of these characteristics is concurrency; i.e., the notion of "independent execution."

In "traditional" process-based designs, the unit of concurrency is the "process". Essentially, a process is the independent executing state of some program, or piece of program, which is conceptually executing in parallel with other processes. In process-based designs, processes interact with other processes,

with responsibilities for the job being accomplished somehow divided between the processes. Generally speaking, a process executes internally, performing some computation, until it must interact with another process before it can proceed. Maybe it needs a piece of information before it can go on, or maybe it has produced a result needed by the other process; the reason doesn't matter. What is important is that the process, for whatever reason, cannot continue its computation until the interaction completes. A process in this condition is said to be "blocked" (from proceeding further), and suspends its execution by calling a primitive system function "block()" [this is the function-call syntax for some arbitrary language]. In theory, the process somehow simply "waits". In reality, by some slight-of-hand, the state of the executing process (its registers, program counter, stack pointer, condition codes, and whatever else dictated by the hardware) is preserved in suspended animation until such time as the necessary interaction is complete. When said interaction completes, a "resume()" function thaws the frozen process which can then continue with its computation.

Nowhere have we explained how such interactions are orchestrated, or who calls "resume()" to bring the sleeping process back to life. This isn't particularly important at this juncture (see the texts for the myriad ways such matters can be conducted). What is important is the model - processes compute and communicate, compute and communicate, in an endless cycle. Moreover, to the frozen process, the block()/resume() sequence appears like a "no-op" instruction (unless a process has access to some external timing information, which makes block()/resume() appear like a very slow no-op!). This style of concurrency promotes programs with multiple processes which either compute or wait on one another to communicate, with the sequencing of those events controlled almost entirely within the individual processes.

One important implication of this structure has to do with the state-saving slight-of-hand described above, and its implementation on single processors. Since a real (single) CPU can only execute one instruction stream at a time, the CPU must be multiplexed between the processes which are logically executing concurrently. The block()/resume() pair form the basis of this multiplexing. If only one of the processes is runnable ("not waiting") at any given time, the choice of which process to run is easy. When several are runnable (the overwhelming case), policies establish how the CPU cycles are to be divided. This is called the "scheduling policy," and it too is discussed at great length in the cited texts. For our purposes, the important issue is whether the "which process to execute" decision is made only when the running process block()'s (thereby voluntarily giving up the CPU), or whether by some magic (usually related to some kind of real-time clock) "the scheduler giveth and taketh away" the processor arbitrarily. This consideration is important because this policy controls the apparent relative rates of execution of the processes. For communications purposes, this controls the "granularity" of the concurrency. [For instance, if programs are allowed to run to completion before the CPU is switched to another process, then the system would essentially not multiprogram.] The actual process of transferring control of the CPU between processes is called a "context switch." Context switches are surprisingly expensive operations, usually 5 to 20 times more expensive than a full-blown subroutine call (a subroutine call is not merely a "jump to subroutine" instruction, but includes passing arguments, etc.).

By now the reader should have a general idea of the basic mechanisms inside a multiprogrammed operating system. The reason for explaining all this is that the Hub is different.

An Architectural Overview of the Hub

This section will describe the basic structure and mechanisms of the Hub using analogy with modern microprocessor architecture. The next section will describe the implications of these mechanisms and contrast them with the traditional process model.

The classic fetch-execute cycle found at the heart of almost all processors (micro or otherwise) is also the basic mechanism of the Hub. The Hub system (Figure 1) takes its name from the Hub Queue which stores unexecuted instructions. The Hub queue is essentially the "instruction prefetch queue" for the "execution unit" of the Hub system. An important feature of the Hub, however, is the source of instructions stored in the Hub queue. In most familiar processors, the instructions to be executed ("the program") are stored in a memory, and they are sequenced into the instruction prefetch queue using the value of "the program counter." Conventionally, the program counter simply increments, causing the instruction at the "next" memory location to enter the prefetch queue. The only exception to this linear sequencing is when a "branch" instruction loads the program counter with some arbitrary new value.

In the case of the Hub queue, there is no static "program" of instructions waiting to be sequenced from memory into the Hub queue by some "program counter." Rather, instructions are created and sequenced into the Hub queue by the execution of other instructions! This means that most (but not necessarily all) instructions generate at least one new instruction when they execute. From this viewpoint, the "program" is created as it executes! Like branches in microprocessors, there is one special case of loading the queue: initializing the system. Some agent must initialize the Hub queue with instructions before the Hub fetch-execute cycle can be engaged, lest there be nothing to do!

With this basic understanding, Figure 1 should be more comprehensible, but there are some additional twists. Modern processors often have partitioned execution units - floating-point add-on chips which interpret special arithmetic instructions are a good example. When an instruction comes to the front of the prefetch queue, the instruction decoder inspects the candidate instruction and dispatches it to the appropriate functional execution unit for interpretation. A similar operation takes place when the Hub dispatches an instruction. In the Hub, an execution unit which can process instructions is called a TASK. (My apologies to readers who already have a favorite interpretation for this and several other words to follow - there are only so many such words which are short and euphonious.) In the Hub system, the Task is the fundamental mechanism for organizing an activity and providing concurrency. A Task is represented by a data area called a TASK STATE VECTOR (TSV) and represents the "register set" of an execution unit. Further, a Task must have some specification of the work it is performing, so a Task is said to be obeying a particular TASK PROGRAM (TP). The Task Program is essentially the "microcode" for the execution unit. Formally, a Task is a binding of a TSV to a TP. Such bindings are not arbitrary as a TP expects a particular data layout in the work area of an associated TSV, but many TSV's can be bound to one TP (even if the TP code isn't truly reentrant). This is like having more than one floating-point register set in a floating-point unit. Both a many-to-one and one-to-one TSV-TP binding is indicated in Figure 1. Two questions are raised by Figure 1: If a TSV is a data area, why do arrows labeled "procedure calls" point to TSV's, and how is the binding between a TSV and its associated TP represented? To address this question, it is valuable to look at Hub instructions in a bit more detail.

Figure 2 is a simplified Hub instruction. At the outset it appears quite familiar - an opcode and a few operands of some fashion. But note the added fields: the source and destination TSVid's. A TSVid is a handle, represented as a small integer, which can be used as a shorthand to indicate a particular TSV, and therefore, a particular Task. (The TSVid is returned to the proud parent when a new Task (TSV) is created.)

When the fetch-execute dispatcher examines the instruction at the front of the Hub queue, it uses the Destination TSVid in the instruction to determine which execution unit (TSV) should process the instruction. The opcode field then specifies which operation is to be performed by that particular execution unit. From the

processor analogy viewpoint, we have simply split the opcode into two pieces for easy decoding: one part specifies which CPU chip should be used for the instruction (floating-point, decimal, normal integer), and the opcode is then private to the particular chip. In the Hub system, however, the Opcode field is at least partially specified across all Tasks: a specific, small set of instructions is implemented (or at least treated rationally) by ALL Tasks. Tasks can also have private instructions in addition to this basic core set, but usually the core set provides all that are needed.

We now return to the question of binding-the TSV with a particular TP and how instructions actually get dispatched. Like the common Opcode set, the first chunk of all TSV's has a fixed format. This area with a common format is called the "TSV Prefix". One of the key items in the TSV prefix is the TASK PROGRAM ENTRYPOINT (TPE) dispatch table. The TPE dispatch table is indexed by the Opcode value and contains a pointer to the function within the TP code body which performs the interpretation of the corresponding Opcode. These function pointers form the binding between the TSV and the TP. This indirection through the TPE dispatch table in the TSV Prefix is why the "procedure call" arrows in Figure 1 go through the TSV's. To tie all the pieces together, the main fetch-execute loop of the Hub Operating System is essentially:

```
Forever {
        Fetch the next instruction from
            the Hub Queue

        Use the Instruction's Destination
            TSVid to locate the TSV which
            is to execute the instruction

        Use the Instruction's Opcode to
            retrieve the appropriate Task
            Program Entrypoint function
            address

        Call the TP function at the
            specified address to actually
            execute the instruction
}
```

Architectural Implications

The Dispatcher loop described above is really all there is to the Hub. There is no notion of "blocking". Task Program Entrypoint functions run to completion and then return to the Dispatcher loop. In the process model, the state of an activity's computation is implicitly contained in the variables (registers, storage, stack segment, etc.) of the process encapsulating the activity, and this thinly-diffused state information must be maintained inviolate by the context switch mechanism. In the Hub, the TSV represents all the storage needed by an activity. Any state information which must be retained across TPE invocations must be recorded in the TSV.

There is only one stack segment in the Hub; the Dispatcher invokes the TPE functions with subroutine calls, so no context switch is needed. (This also means no tricky assembler code is required, since context switch functions are almost always written in assembler.) Further, in process systems, processes often "idle" internally when they have nothing to do. In the Hub, a TSV with nothing to do doesn't get dispatched. If the Hub queue runs out of something to do, the Dispatcher idles waiting for an instruction. Where do these instructions come from? Interrupts.

In most systems, interrupts are viewed as nasty creatures which are to be sterilized as quickly as possible. In process-based designs, the usual litany is to make the interrupt simply awaken a sleeping process which then does whatever is necessary to placate the device. In systems with high interrupt rates, this means high context-switching rates, and poor performance.

In the Hub environment, interrupt code usually does some simple work to clean up after the device, and start a new operation if there is an outstanding queue. Notification of the client activity that the operation has completed is accomplished by simply queuing an instruction. Therefore, if interrupts are enabled and devices are running, it is allowable for the Hub queue to empty. The system is simply waiting on I/O. One other source of interrupts is the real-time clock. This is implemented by wiring a periodic source to an interrupt input so as to produce a periodic interrupt heartbeat at a known rate (usually 60-100 Hz). In communications systems, the situation often arises where there is no other I/O pending, but the system is idling, waiting for some time to pass before some Task starts dumping a retransmit queue, etc. The clock interrupt provides the timebase which drives the Timer Management functions, which in turn provide timer services to TSVs.

Hub Support Services

In addition to providing concurrency, an operating system also provides support services for activities. Memory management is an important service, as are time management, buffer, and queue management. These services provide useful primitives which need not be reengineered by each Task, and are orchestrated in such a manner as to promote cooperation between Tasks.

Memory and Buffer Management

In this Hub implementation, memory management and buffer management have been lumped together. They can easily be split back apart (even in this implementation) but using the same allocator for buffers and what would normally be "general memory requests" is very attractive. The overwhelming experience with communications programming is that internal fragmentation induced by fixed size memory allocation is much easier to tolerate than the external fragmentation which results from variable size memory allocation. Further, fixed-size allocators are always faster, usually by a large margin, since they don't have to look around to find a piece of memory big enough to satisfy the request, nor try to coalesce chunks when they are freed. Two problems arise with fixed-size allocators: (1) the small-object/large-object problem, and as a result of addressing(1), we find (2) a maximum allocation limitation.

Network traffic comes in two sizes - small packets containing terminal traffic, and large packets containing file transfer data. If storage is allocated in chunks big enough for the large packets, much storage will be wasted. If we chose to allocate small chunks, we must insure the overhead of keeping track of them doesn't consume too much storage or processor bandwidth. The compromise is to allocate storage in medium-sized units.

The storage allocation units are sized to accommodate some large percentage of all requests (ranked by size and frequency). When large packets are required, the buffer header contains pointers which can be used to chain buffers together to form multi-buffer "messages", in addition to the usual pointers for chaining buffers into a queue of distinct messages.

A direct implication of a small/large compromise allocation size, is a rather severe upper bound on the size of an object which can be created using storage acquired from the allocator. In a general-purpose programming environment, this would be unacceptable. For communications programming, however? this doesn't usually pose a hardship, although it occasionally complicates some algorithms a bit more than they might be in other circumstances. If future applications of this Hub implementation find themselves in hardship situations, as was stated above, the memory and buffer allocator can be split apart to provide the necessary level of service.

Clock and Timer Management

In this implementation of the Hub architecture, we assume a single, fixed-period real-time interrupt which is vectored (somehow!) to the Clock Interrupt Handler. One interrupt is called a "tick", and represents the smallest real-time increment which can be directly measured or created in a given implementation. The Clock and Timer support routines provide Task Programs with

functions for scheduling timer notification events after a specified interval, canceling scheduled events, measuring real-time intervals, and maintaining the current time-of-year.

The Timer functions maintain a queue of upcoming timer events with the queue elements recording the difference between the time of their event and the previous event, measured in ticks. This way, the clock interrupt code need only decrement the time remaining in the first element of the timer event queue until it is zero. When the difference has been reduced to zero, the appropriate TIMER instruction destined for the recipient is placed in the Hub queue, and the timer queue element recycled. This process of "decrement once per tick until a zero difference is detected" repeats as long as there are timer queue entries.

This does place a burden on the queuing functions. The timer queue element must be sorted into the queue in the correct location, while adjusting the requested absolute time to the appropriate difference while scanning the current queue. When a scheduled event is deleted from the queue, only the following element must be adjusted. The algorithms are not complex, but do require careful attention to the boundary conditions.

Queue Management

In this implementation, queues are doubly-linked and are headed with a dummy queue cell. This affords easy insertion, deletion, and sanity-checking. The usual assortment of insert, delete, and discard functions are provided. They are optimized for queues of buffers.

Unexplored Issues

This paper, while providing an overview of the Hub (and some background for the operating systems newcomer), has left some important issues unaddressed. While the "CPU" model of instruction execution spawning new instructions which are added to the Hub Queue is fairly straightforward to grasp, it offers only modest direct insight as to how one structures concurrent systems using the Hub primitives. The actual subroutine calls necessary to do the work in a Hub System, and how one goes about rehosting a Hub implementation are also important issues which weren't addressed either. But this paper only claimed to be an Introduction to the Hub Operating System, so some these issues remain ideal topics for other documents, and others are best addressed by reading the code. We intend to report our experiences in all these areas.

Bibliography

Comer, D., "Operating System Design: the XINU Approach," Prentice-Hall, 1984

Jansen, P.A., "Operating Systems Structures and Mechanisms," Academic Press, 1985

Hansen, B., "Operating Systems Principles," Prentice-Hall, 1973

Masamoto, K., "Implementation of HUB System," University of Illinois Master's Thesis, 1976

Organick, E.I., "Computer Systems Organization: The B5700/B6700," Academic Press, 1973

Organick, E.I., "The Multics System,", MIT Press, 1972

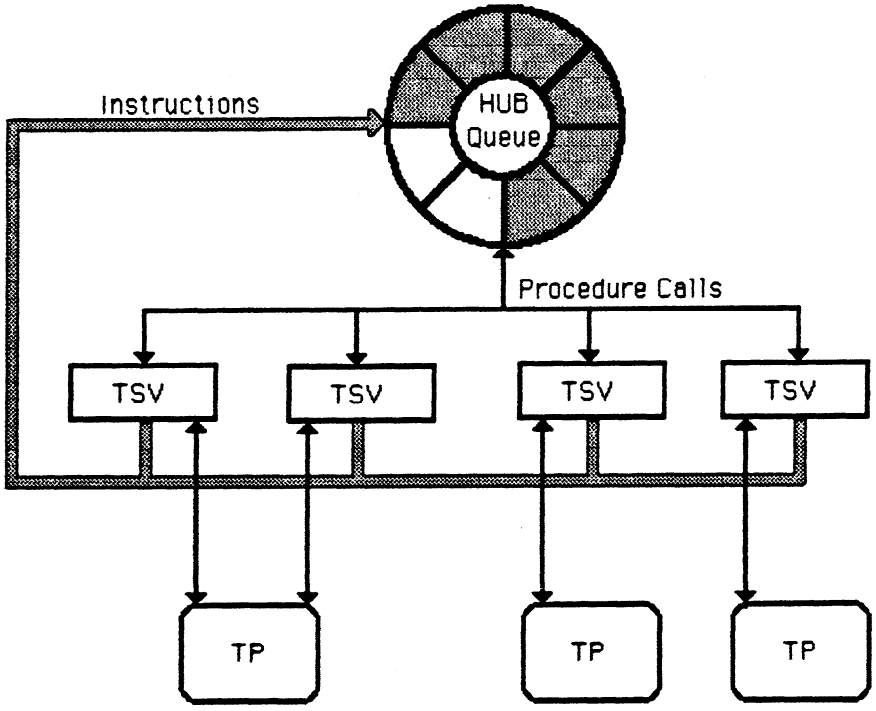


Figure 1
The HUB System

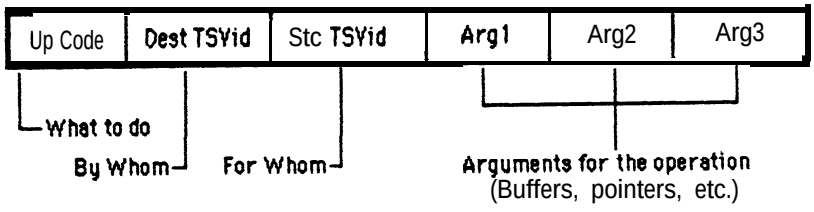


Figure 2
A Hub Instruction

PROPOSAL: RECOMMENDATION AX.224, CLASS 1

TRANSPORT PROTOCOL SPECIFICATION FOR THE AMATEUR RADIO NETWORK

J. Gordon Beattie, Jr., N2DSY
Thomas A. Moulton, W2VY

The Radio Amateur Telecommunications Society
206 North Vivyen Street
Bergenfield, NJ 07621
201-387-8896

Introduction

The amateur packet network provides a reliable service in that it is relatively free from undetected bit errors. It does, however, have a relatively high rate of lost connections. This problem can be resolved through the implementation of a TRANSPORT PROTOCOL. This is not to suggest that all applications require a TRANSPORT PROTOCOL. In situations where end-to-end data and connection integrity are important, one must use a TRANSPORT PROTOCOL to provide error control.

Recognizing that there are many options available to the community, we the RADIO AMATEUR TELECOMMUNICATIONS SOCIETY, felt that there had to be a single defined protocol available to the broadest possible user base. To achieve this we examined several protocols and determined that the one most appropriate for amateur service was CCITT Recommendation X.224, Class 1. This protocol was chosen for its applicability, simplicity, expandability, and international acceptance.

It is hereby proposed by the members of the RADIO AMATEUR TELECOMMUNICATIONS SOCIETY that this basic subset of CCITT Recommendation X.224 be adopted by the amateur packet community as the preferred transport protocol.

1. SCOPE

1.1 The protocol will detect lost packets not detected by the Network Layer and institute required error recovery.

1.2 The protocol will detect the loss of the underlying network connection and institute required error recovery.

2. DEFINITIONS

Data Units

TSDU - Transport Service Data Unit
This is the basic data unit which requires transmission and acknowledgement. Each TSDU may be segmented into many DT-TPDUs and sent across the network.

TPDU - Transport Protocol Data Unit
This is the fundamental unit that is used to convey control information and data between TS-users. TPDUs may not exceed the maximum NSDU length provided by the network.

NSDU - Nework Service Data Unit
This is the data unit used by the underlying network to transmit information.

TPDU Types

CR - Connection Request
CC - Connection Confirm
DR - Disconnect Request
DC - Disconnect Confirm
DT - Data
AK - Data Acknowledge
RJ - Reject
ER - Error

Other

LI - Length Indicator
CDT - Credit
TSAP-ID - Transport Service Access Point Identifier
DST-REF - Destination Reference
SRC-REF - Source Reference
EOT - End of TSDU Mark
TPDU-NR - DT TPDU Number
TSAP - Transport Service Access Point

3. TRANSPORT LAYER FUNCTIONS

3.1 Connection Establishment

The purpose of connection establishment is to establish a transport connection between two transport service (TS) users. The CR-TPDU is sent by the originating TS-user and the request is confirmed by the reply of a CC-TPDU from the correspondent TS-user.

If the connection is not possible, a DR-TPDU may be sent in reply to the CR-TPDU. The originating TS-user would then confirm this rejection by the transmission of a DC-TPDU.

References are under local control, but in order to ensure that data is properly handled by the TS-provider, they should not be re-used until the list is exhausted. In the event of a network failure that prevents reassignment of the transport connection, the references should be frozen for an extended period. The exact duration is a local issue.

During the establishment phase certain parameters may be conveyed and negotiated with the correspondent TS-user. These parameters are outlined in section 6.

3.2 Data Transfer

The data transfer phase permits the TS-users the use of a full-duplex transmission path. The DT-TPDU is used to convey TS-user data across the network. A single TSDU may be segmented into several DT-TPDUs. All TSDUs or DT-TPDU sequences are explicitly acknowledged through the use of the AK-TPDU. A DT-TPDU sequence is completed through the setting of the EOT-bit in the last DT-TPDU.

3.3 Release

The connection release phase allows for the disconnection of a transport connection. It is signalled by the transmission of a DR-TPDU and confirmed by the reception of a DC-TPDU from the correspondent TS-user.

3.4 Resynchronization

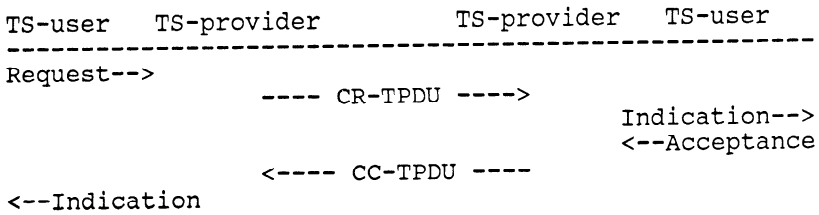
If a TDPU in a DT-TPDU sequence is lost due to network reset or error, the receiving TS-user will reply with the transmission of a RJ-TPDU. This RJ-TPDU will provide the correspondent TS-user with an indication of which DT-TPDU was lost.

3.5 Reassignment

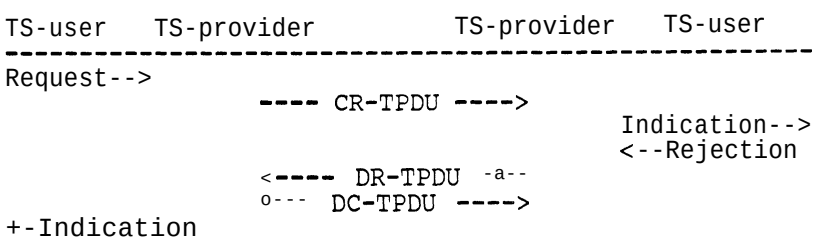
This capability allows a transport connection to recover from a signalled disconnect in the underlying network service. When this occurs, the TTR timer should be started and the transport connection should be reassigned to a new network connection. If TTR expires, the reference should be frozen and the transport connection should be released.

4. TRANSPORT LAYER PROCEDURES

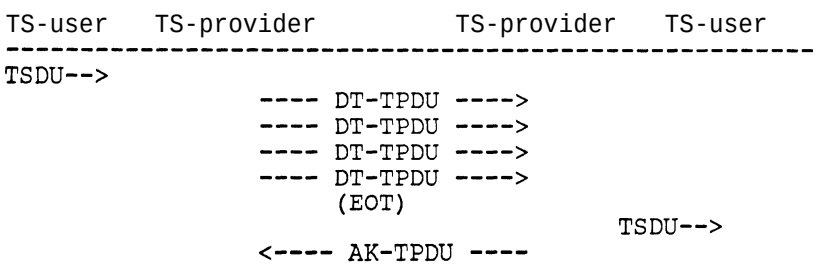
4.1 Connection Establishment



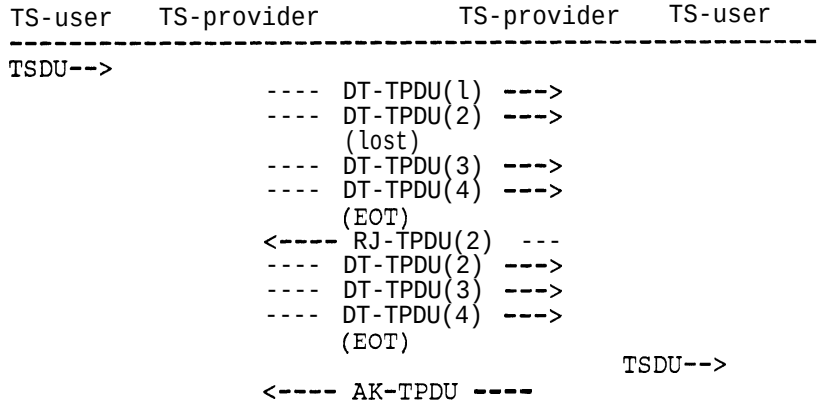
4.2 Connection Rejection



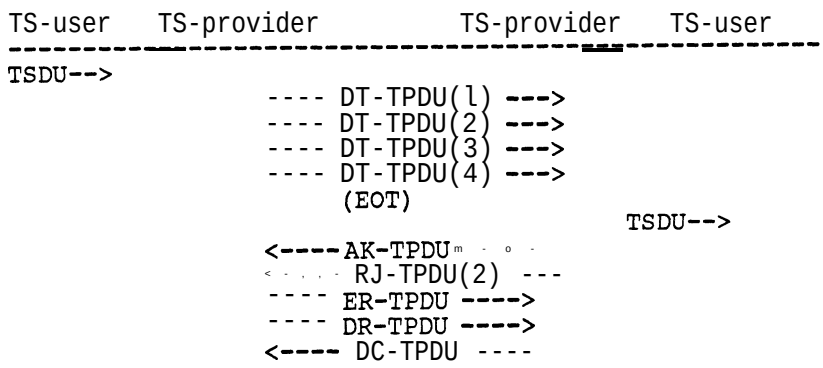
4.3 Data Transfer



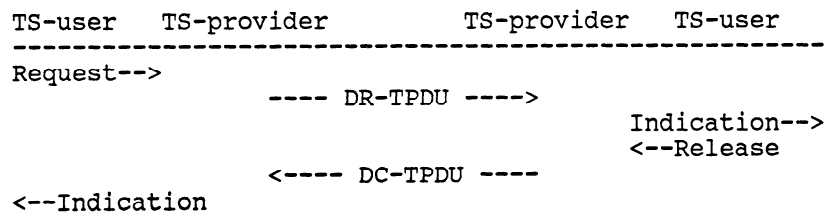
4.4 Reject



4.5 Protocol Error



4.6 Connection Release (normal)



5. STRUCTURE AND CODING OF TPDUs

5.1 Validity

Table 5/AX.224 specifies those TPDU's which are valid for class 1 and the code for each TPDU.

TABLE 5/AX.224

CR	Connection Request	**11100000**	**5.3**
cc	Connection Confirm	**11010000**	**5.4**
DR	Disconnect Request	**10000000**	**5.5**
DC	Disconnect Confirm	**11000000**	**5.6**
DT	Data	**11110000**	**5.7**
AK	Data Acknowledge	**01101111**	**5.8**
RJ	Reject	**01011111**	**5.9**
ER	TPDU Error	**01110000**	**5.10**

5.2 Structure

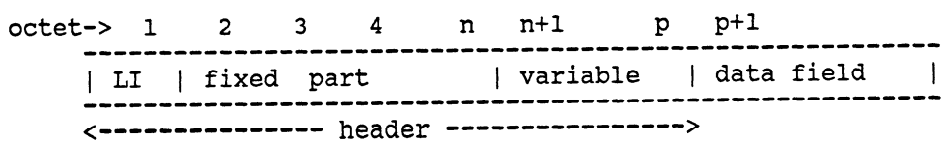
All the TPDU's shall contain an integral number of octets. The octets in a TPDU are numbered starting from 1 and increasing in the order they are in into the Network Service Data Unit (NSDU). The bits in an octet are numbered from 1 to 8, where bit 1 is the low-order bit.

When consecutive octets are used to represent a binary number or a binary coded decimal number (one digit per octet), the lower number octet has the most significant value.

TPDUs shall contain, in the following order:

a) the header, comprising:
   1) the length indicator (LI) field:
   2) the fixed part:
   3) the variable part, if present:
b) the data field, if present.

This structure is illustrated below



5.2.1 Length Indicator Field

This field is contained in the first octet of the TPDU's. The length is indicated by a binary number, with a maximum value of 254 (1111 1110). The length indicated shall be the header length in octets including parameters, but excluding the length indicator field and user data, if any. The value 255 (1111 1111) is reserved for possible extensions.

If the length indicated exceeds the size of the underlying NSDU this is a protocol error.

5.2.2 Fixed Part

The fixed part contains frequently occurring parameters including the code of the TPDU. The length and the structure of the fixed part are defined by the TPDU code.

If any of the parameters of the fixed part have an invalid value, or if the fixed part cannot be contained within the header (as defined by LI) this is a protocol error.

5.2.3 Variable Part

The variable part is used to define less frequently used parameters. If the variable part is present, it shall contain one or more parameters.

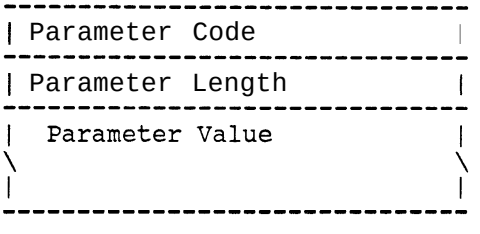
The parameter code field is coded in binary.

The parameter length is limited to 248. In the case of a single parameter contained within the variable part, two octets are required for the parameter code and the parameter length indication itself. For larger fixed parts of of the header and for each succeeding parameter the maximum value decreases.

The parameters defined in the variable part may be in any order.

An invalid parameter will be treated as a protocol error.

Each parameter contained within the variable part is structured xas follows:

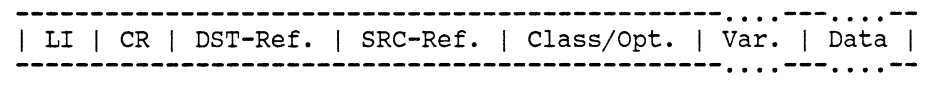


5.2.4 Data Field

This field contains transparent user data. Restrictions on its size are noted for each TPDU.

5.3 Connection Request - CR

The length shall not exceed 128 octets.



CR is set to 11100000B.

The Destination Reference (DST-Ref.) is coded as 0000H.

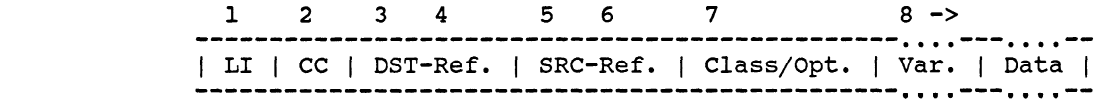
The Source Reference (SRC-Ref.) is selected by the initiating transport entity.

The Class/Options octet is set to 00010000B.

The variable part contains parameters defined in section 6.

The CR may contain up to 32 octets of user data.

5.4 Connection Confirm - CC



CC is set to 11010000B.

The Destination Reference (DST-Ref.) is selected by the remote transport entity.

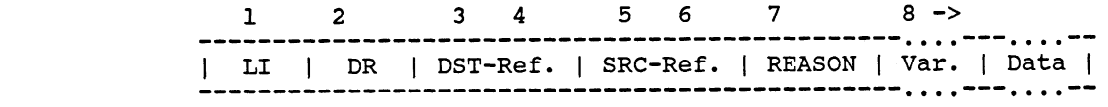
The Source Reference (SRC-Ref.) is selected by the initiating transport entity.

The Class/Options octet is set to 00010000B.

The variable part contains parameters defined in section 6.

The CR may contain up to 32 octets of user data.

5.5 Disconnect Request - DR



DR is set to 10000000B.

The Destination Reference (DST-Ref.) is selected by the remote transport entity.

The Source Reference (SRC-Ref.) is selected by the initiating transport entity.

The Reason code is the reason for disconnection. The values include:

128 + 0	Normal Disconnect
128 + 1	Congestion at Connect Request Time
128 + 2	Connection Negotiation Failed
128 + 3	Duplicate Connection Detected
128 + 4	Mismatched References
128 + 5	Protocol Error
128 + 6	Not Used
128 + 7	Reference Overflow
128 + 8	Connection Request On This Network Connection
128 + 9	Not Used
128 + 10	Header or Parameter Length Invalid

Additional codes include:

0	Reason Not Specified
1	Congestion TSAP
2	Session Entity Not Attached to TSAP
3	Address Unknown

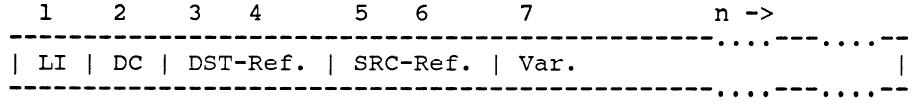
The variable part may contain a parameter with additional information related to the clearing of the connection.

The Parameter Code Value is 11100000B

The parameter length may be any value provided that the DR-TPDU length does not exceed the maximum agreed TPDU size or 128 when the DR-TPDU is used during the connection rejection procedure.

The DR may contain up to 32 octets of user data.

5.6 Disconnect Confirm - DC

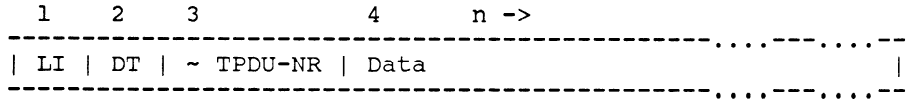


DC is set to 11000000B.

The Destination Reference (DST-Ref.) is selected by the remote transport entity.

The Source Reference (SRC-Ref.) is selected by the initiating transport entity.

5.7 Data - DT



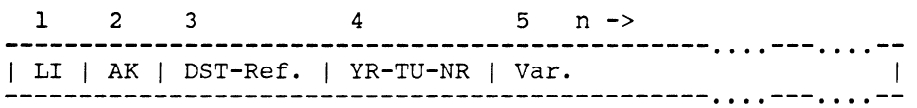
DT is set to 11110000B.

~ The EOT bit, when set to one, signals that this DT-TPDU is the last in a sequence carry a TSDU. This bit is bit 8 of octet 3.

TPDU-NR is the TPDU Send Sequence Number. The TPDU-NR uses bits 7-1 in octet 3.

The data field may contain up to the negotiated TPDU size minus 3 octets. (Header overhead)

5.8 Data Acknowledge - AK

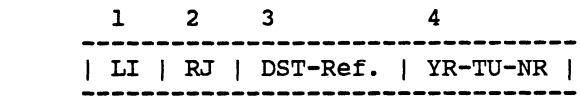


AK is set to 01101111B

The Destination Reference (DST-Ref.) is selected by the remote transport entity.

The YR-TU-NR) is the sequence number indicating the next expected DT-TPDU number. Bits 7-1 shall indicate this value. Bit 8 is not used and shall be set to 0.

5.9 Reject - RJ

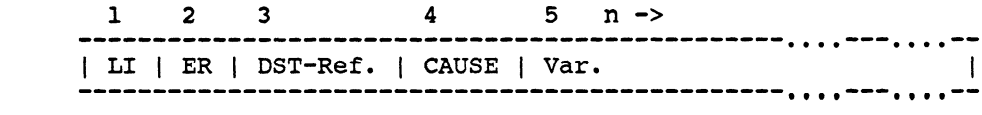


RJ is set to 01011111B

The Destination Reference (DST-Ref.) is selected by the remote transport entity.

The YR-TU-NR) is the sequence number indicating the next expected DT-TPDU number from which retransmission should occur. Bits 7-l shall indicate this value. Bit 8 is not used and shall be set to 0.

5.10 Error - ER



ER is set to 01110000B

The Destination Reference (DST-Ref.) is selected by the remote transport entity.

Reject Cause:

0 Not Specified
1 Invalid Parameter Code
2 Invalid TPDU Type
3 Invalid Parameter Value

The variable field contains the Invalid TPDU parameter.

The parameter code is 11000001B.

The parameter length is the number of octets in the value field.

The parameter value contains the bits pattern of the rejected TPDU header up to and including the octet which caused the rejection.

6. CONNECTION ESTABLISHMENT PARAMETERS

The connection establishment parameters allow the TS-user to select operational characteristics required for the support of the connection. These parameters provide the underlying network service provider with an indication of what facilities are required by this user. These paramters are outlined below.

6.1 TPDU Size

Parameter Code: 11000000B

Parameter Length: 1 octet

Parameter Value:

ODH 8192 octets
OCH 4096 octets
OBH 2048 octets
OAH 1024 octets
09H 512 octets
08H 256 octets
07H 128 octets

Default value: 07H (128 octets)

6.2 Version Number

Parameter Code: 11000100B

Parameter Length: 1 octet

Parameter Value: 00000001B

Default value: 00000001B

6.3 Security Parameters

Parameter Code: 11000101

Parameter Length: user defined

Parameter Value: user defined

6.4 Additional Option Selection

Parameter Code: 11000110

Parameter Length: 1 octet

Parameter Value: 00000000B

Use of this parameter is mandatory.

6.5 Throughput

Parameter Code: 10001001

Parameter Length: 12 or 24 octets

Parameter Value:

1st 12 octets: maximum throughput, as follows:

- 1st 3 octets: target value, calling-called user direction;
- 2nd 3 octets: minimum acceptable, calling-called user direction:
- 3rd 3 octets: target value, called-calling user direction;
- 4th 3 octets: minimum acceptable, called-calling user direction.

2nd 12 octets: average throughput, as follows:

- 5th 3 octets: target value, calling-called user direction;
- 6th 3 octets: minimum acceptable, calling-called user direction;
- 7th 3 octets: target value, called-calling user direction:
- 8th 3 octets: minimum acceptable, called-calling user direction.

Where the average throughput is omitted, it is considered to have the same value as the maximum throughput.

Values are expressed in octets per second.

6.6 Residual Error Rate

Parameter Code: 10000110B

Parameter Length: 3 octets

Parameter Value:

1st octet: Target Value, power of 10
2nd octet: Minimum Acceptable, power of 10
3rd octet: TSDU size of interest, expressed as a power of 2.

6.7 Priority

Parameter Code: 10000111B

Parameter Length: 2 octets

Parameter Value: Integer (High = 0)

6.8 Transit Delay

Parameter Code: 10001000

Parameter Length: 8 octets

Parameter Value:

1st 2 octets: target value, calling-called user direction:
2nd 2 octets: minimum acceptable, calling-called user direction:
3rd 2 octets: target value, called-calling user direction;
4th 2 octets: minimum acceptable, called-calling user direction.

Values are expressed in miliseconds, and are based upon a TSDU size of 128 octets.

6.9 Reassignment Time

This parameter conveys the Time to Try Reassignment / Resynchronization (TTR) which will be used when following the procedure for Reassignment after Failure.

Parameter Code: 10001011

Parameter Length: 2

Parameter Value: n, a binary number, where n is the TTR value expressed in seconds.

CRYPTOGRAPHY IN AMATEUR RADIO COMMUNICATIONS

Robert M. Richardson, W4UCH
22 North Lake Drive
Chautauqua Lake, N.Y. 14722

ABSTRACT:

Some fascinating similarities between the art and science of cryptography and the amateur radio avocation, especially in the area of digital communications are discussed.

GENERAL:

Webster's New World Dictionary defines cryptography as: "The art of writing or deciphering messages in code." As such, every U.S. radio amateur is a cryptographer, some willingly and some kicking and screaming. Even the most erstwhile novice or technician class radio amateur is a cryptographer when he studies for the 5 words per minute Morse code examination. Some radio amateurs become so proficient at deciphering Morse that they can carry on a conversation while copying Morse on a typewriter at 20 - 30 words per minute. Some radio amateurs once they pass the code test never want to hear Morse code again. Either approach is perfectly acceptable within the radio amateur fraternity as it is a house of many mansions with room for all kinds with differing interests and differing specialties.

Let's take a look at some of the synonyms used in the cryptography field before we begin to dig deeper. To code a message, regardless of the variety of coding used, we may use any of the following terms in the English language as they all mean virtually the same thing: encrypt, encode, encipher, or in slang, even the term scramble. The opposite function which returns a message to plaintext is of course: decrypting, decoding, deciphering, or descrambling.

BAUDOT CODE:

Moving up the ladder of commonly used amateur radio encryption we have Baudot radioteletype (RTTY) which has roots back to about 1906 in radio prehistory. It is a synchronous 5 bit code. By synchronous we mean that is has start and stop bits in addition to the 5 data bit code. Having a start and stop bit, we may send Baudot RTTY at say 60 speed and if we are a hunt a peck typist, we might only transmit 10 or so words per minute even though each character is transmitted at a 60 speed rate.

ASCII CODE:

The next level of amateur radio RTTY encryption uses the American National Standard Code for Information Interchange, ASCII. It too is a synchronous code with start and stop bits in addition to its 7 data bits and optional 8th parity bit. On the low frequency bands commonly used Baud rates are 110 and 300 with an -equivalent speed in words per minute of 110 and 300 if the transmission is being sent automatically.

PACKET:

On up the amateur radio encryption ladder another step we find our relatively new friend, packet, first implemented on the ham bands by radio amateurs Rouleau and Hodgson in Quebec circa 1979. This relatively new form is the first synchronous amateur radio data communication system. BY synchronous we mean that no start and stop bits are transmitted, only a stream of ones and zeros in each packet. This system was invented in 1957 by IBM's brilliant Robert Donan and was initially called Synchronous Data Link Control, SDLC.

IBM SDLC:

Defining the fundamental concepts of SDLC in one paragraph is a challenge, but here goes. SDLC, those parts used by radio amateurs is an 8 data bit per byte encrypted code. Each packet (or frame) has a unique opening and closing flag byte or bytes that never appear elsewhere in the packet. Between these flags a logical zero is noted by a change from the previous bit received; i.e., if it changed it is a logical zero. If it did not change it is a logical one. In addition, SDLC uses 'zero insertion' between flags to keep the unique flag byte (126 decimal) from ever being repeated, except when it is used to mark a packet (or frame) opening and closing boundary. The SDLC rule is: whenever more than 5 logical ones are to be transmitted between flags, insert a zero = 'zero insertion.' On receiving, 'zero deletion'

is used; i.e., when a logical zero is received after 5 logical ones, delete it. Before the closing flag is sent in a packet (or frame) a 2 byte cyclic redundancy check (CRC) value is calculated and transmitted. Packets may be generated and decoded either by software or hardware depending upon which variety of packet controller you are using. Commonly used packet Baud rates on the VHF bands are 1200 Baud and up, and on the HF bands, 300 Baud.

Ax. 25 PROTOCOL:

Encoding and decoding packets in real time using the software approach is a fascinating challenge for the amateur radio cryptographer. First, the SDLC discipline must be thoroughly understood. Second, generating and/or checking a real time CRC must be understod. Third and last, the protocol being used must be understood. The latter is probably the most challenging, but with the excellent ARRL book delineating the AX.25 protocol is not all that difficult.

DIGITAL AUDIO ENCODING:

The newest kid on the block, after companded SSB, is digital audio. It will come to pass for the amateur radio fraternity in due course. The' single sideband enthusiasts will scream just as loudly as the AM enthusiasts screamed back in the 1950s when SSB was introduced. It makes no never mind as technology progresses with or without the diehards' approval or disapproval.

CRYSTAL BALL GAZING:

Looking into our crystal ball we can see the early digital audio transmission modes using frequency shift keying (FSK) and phase shift keying (PSK) giving way to more efficient binary phase shift keying (BPSK) and quadrature phase shift keying (QPSK), If a radio amateur cryptographer wishes to get his or her feet wet in the digital audio field, the C-band and Ku-band geosynchronous satellites are a good place to start as most every variety of digital audio is being used. A great deal of the digital audio traffic is just plaintext, with no added encryption of any variety being used.

For those radio amateur crytographers who would like a real challenge, there is now encrypted digital audio on the C-band Galaxy I satellite located at 334 degrees west in the Clarke belt. It may be found on channel 19 with a downlink frequency of 4080 MHz and channel 23 with a downlink frequency of 4160 MHz, both with horizontal polarization. Both channels of audio are encrypted using a modified form of the 56 bit key digital encryption standard (DES).

Is it impossible to decipher a DES encrypted digital audio signal? Of course not. It may be difficult, but it is not impossible. Some radio amateur cryptographers have estimated that it would take a dozen super duper Cray maxi-computers 29 days to search through all 72 quadrillion possible keys to decrypt this type of encoded digital audio.
WRONG: They have overlooked two important facets of the problem at hand:
3. The key (though encoded) is transmitted over the air and may be recorded along with the digital audio signal on any video recorder.
2. The plaintext of the encoded digital audio is available to be recorded from any satellite TV receiver with a decoder.

IS IT LEGAL TO RECORD ? ? ?

Of course it is. The master recordings above were made by a paid subscriber to these services.

DES USERS GROUP:

If you are an amateur radio cryptographer and would like to join the DES Users Group that is investigating this fascinating challenge, send a S.A.S.E. to the DES Users Group, Drawer 1065, Chautauqua, NY 14722, for an info sheet. There are no dues at present. The only requirement is an active interest in amateur cryptography. The 1st annual meeting will be held at the Dayton Hamfest, April 25 - 27, 1986.

THE UO-11 DCE MESSAGE STORE-AND-FORWARD SYSTEM

Harold E. Price, NK6K
1211 Ford Ave
Redondo Beach, CA 90278

Jeff Ward, GO/K8KA
Dept. of Electrical Engineering
University of Surrey
Guildford, Surrey GU2 5XH
England

Abstract

The Digital Communications Experiment (DCE) onboard the UoSAT-Oscar-11 spacecraft recently began a new phase of regular operations. Development and installation of enhanced store-and-forward message transfer software (MSG2) - capable of 200-kbytes transatlantic data transfer per day - is the second plateau in the DCE experimental program. This program is designed to gain experience with computer-based message systems in low earth orbit.

The DCE is the first orbiting store-and-forward device to carry general amateur traffic on a continuing basis. The drafts for this paper were developed and edited by the collaborators in the USA and the UK using the spacecraft as the only means of communications.

This paper provides information on the capabilities and the design of this system as well as some background information on the UoSAT-OSCAR 11 spacecraft.

1.0 BACKGROUND

The UO-11 spacecraft, also known as UoSAT-2, was designed and built at the University of Surrey in England during the second half of 1983. It was known as UoSAT-B until its launch from Vandenberg Air Force Base near Lompoc California in March, 1984.

The possibility of flying a small store-and-forward message experiment onboard UO-11 was first discussed at a PACSAT design meeting in July 1983. Amateur groups in Dallas, Los Angeles, Ottawa, and Tucson immediately began work. A flight ready unit was turned over to the integration team at Surrey five months later, A partial account of this whirlwind development can be found in AMSAT's "Orbit" magazine number 18, March/April 84.

1.1 The Spacecraft

The uo-11 spacecraft is a cuboid with dimensions of 35.5cm x 35.5cm x 58.5cm. There are solar cells on the four long faces, generating a total of 35 watts of power, which is stored in a 6-amp-hour, NiCd battery. Included in the complement of experiments are:

- CCD camera with 384 x 256 pixel image capability with 128 levels of gray scale.
- Three particle detectors (Geiger counters) and multi-channel electron spectrometer.
- Space dust (micrometeorite) detector.
- Magnetometer - to measure magnetic field and to determine spacecraft attitude.
- 192kbytes CMOS memory for storage of CCD camera and particle/wave data.

The spacecraft has three downlinks: 145.825, 435.025, and 2401.5 MHz. The 145 MHz downlink is usually on. The 435 MHz downlink is now regularly used for DCE operations. The 2.4 Ghz downlink is rarely used.

The main spacecraft control computer - the On Board Computer (OBC) - is based on an 1802 microprocessor with 48kbytes of static RAM.

1.2 DCE Hardware

The Digital Communications Experiment (DCE) is an important experiment on uo-11 -- establishing that store-and-forward communications in low-earth orbit is realizable and thus fulfilling one of the UO-11 mission objectives. The major goal of the DCE is to provide a software and hardware testbed for PACSAT-type store-and-forward devices. To that end, it was designed to be as flexible as possible. The DCE is all CMOS and contains:

- An NSC-800 CMOS microprocessor using the 280 instruction set.
- Two 2 Harris HD-6402 UARTs.
- One 82C55 parallel port
- 14k of 2kx8 static RAM, Harris 6516.
- 16k of Harris 6564 16kx4 static RAM, using 12 bits to store 8 with single bit error detection and correction in hardware.
- 64k of 8kx8 static RAM, Hitachi 6264LP.
- 32k of 2kx8 static RAM, Hitachi 6116L.

- 512 bytes of Harris 6641 bootload PROM. This PROM has an identical, command selectable backup device.

- command selectable clock speeds of .9 and 1.8 MHz.

The DCE resides on three circuit boards which fit in a standard UoSAT module box approximately 6" x 9" x 1". It draws 120ma at +5 volts.

The total DCE memory capacity is 126kbytes, bringing the total memory aboard UO-11 to 366k -- far exceeding the total memory previously flown by amateur spacecraft.

1.3 Early DCE Operations

UO-11 DCE began its orbital operations on June 5, 1984. Initially, it supported spacecraft operations -- this unplanned activity made necessary by the post-launch failure in an uplink data communications path. UoSAT spacecraft have considerable redundancy in this area, and the problem could be bypassed by routing all VHF spacecraft communication through either the DCE computer or the spacecraft's main onboard computer (OBC). The DCE provided this bypass function in the initial months of spacecraft operations. Since that time, the software in both computers has matured sufficiently to perform the command bypass function while carrying out their other duties. The OBC carries out autonomous operational control of the spacecraft and its experiments, and also automatically determines and adjusts the satellite's attitude. The DCE is dedicated to the message store-and-forward function.

A prototype message system, developed by Hugh Pett, VE3FLL, was used for a demonstration of low earth orbit store-and-forward capabilities at the Pacific Telecommunication Conference in Hawaii in January 1985. The demonstration was done by Hugh and Larry Kayser, WA3ZIA, with support by Harold Price, NK6K and Chris Wachs, WA2KDL in Los Angeles; and Martin Sweeting, G3YJO and the UoSAT team in Surrey.

1.4 Current DCE Activities.

MSG2, the current DCE software, was developed in November 1985 by Harold Price, NK6K, and Jeff Ward, K8KA, at the University of Surrey's UoSAT laboratory. Assistance in implementing the MSG2 ground segment on the BBC micro was provided at UoS by Michael Meerman, PA3BHF. The spacecraft automated control software, DIARY, which also permits DCE ground stations to command the downlinks and multiplexors, was written by Steve Holder (UoS).

2.0 MSG2 SOFTWARE

MSG2 supports the following features:

- Stores up to 96k bytes of message data.

- A single message can be up to 16k bytes.

- Up to 128 messages may be stored at one time.

- A partially downlinked message can be continued at a later time without repeating parts of the message already received.

- A partially uplinked message can be continued at a later time without retransmitting parts of the message already sent.

- If a ground station looses positive control, the DCE will automatically revert to a known state after 2 minutes. The spacecraft DIARY program will also return the downlinks and data multiplexors to a known state after 15 minutes.

- The MSG2 protocol provides complete data transparency.

- The ground station's transmit/receive changeover time is not a factor in communications. The ground station can be full duplex, half duplex using computer controlled (fast) switching, or half duplex using human (slow) switching.

Restrictions in this version:

- Data transfer is in one direction at a time. Acknowledgments can be full duplex.

- Only one ground station can interact with the DCE at one time. The MSG2 software provides the means to keep ground stations from accidentally violating the restriction.

- The integrity of message data stored is not currently guaranteed as the message storage area of the DCE memory is not protected against externally induced errors. The program and non-message data are protected by hardware.

These restrictions may be lifted in later implementations.

MSG2 consists of three elements, a protocol specification, the MSG2 software running on the DCE, and several implementations of software for various computers which implement the MSG2 protocol for ground users.

2.1 MSG2 Protocol

The MSG2 protocol was design primarily to be easy to implement. Its only other goal was to provide the minimal message handling capability to PUT a message on the DCE, to GET one back, and to KILL a message no

longer required.

As "easy to implement" dictated a single user approach, a LOGON and LOGOFF capability was added to keep two or more ground stations from starting message transfer operations simultaneously.

Experimentation with minimal ground stations is planned; the MSG2 protocol was designed to accommodate this activity. Messages are broken into small (64 byte) blocks with CRC error detection. Once a message transfer is begun, message blocks can be acknowledged at any time, and in any quantity. This allows a battery powered station to reduce its transmissions by requiring only one acknowledgment for a message of any arbitrary number of blocks. Unacknowledged blocks are retransmitted in a "round robin" fashion.

DCE blocks are acknowledged by sending a bit map frame. The bit map contains one bit for each block in a message. Bits set to 1 represent unacknowledged blocks, and 0s represent acknowledged blocks (Fig 1). The transmitting station continues to send the blocks indicated by 1 bits, until a bitmap is received with all bits set to 0.

Figure 1. -- Example of an MSG2 Bit Map

```
7 6 5 4  3 2 1 0     (1)

0 0 1 0 1 0 0 0      (2)

0 1 2 3 4 5 6 7      (3)
```

(1) numbering of bits in bit map (MSB is 7)
(2) bit map ack'ing all but blocks 2 and 4
(3) blocks represented by bit map bits

2.2 MSG2 Frame Format

This section is not meant to provide a formal MSG2 protocol specification, but to outline the structure of the protocol and the frames used by it. Frame types may be added or removed as the protocol matures.

Although there are several types of frames, they all share the following format:

<10h><03h><cmd><cmd not><data length><data><crc>

Each byte is sent as an asynchronous character with 8 data bits and no parity bit. Frames are preceded by several SYN bytes <16h> for modem and timing synchronization.

Frame breakdown:

<cmd> -- A single ASCII character specifying a DCE command.

<cmd not> -- The inverse of <cmd>. This byte can be calculated by <CMD> XOR FFh or by 255 minus <cmd>.

<data length> -- A single byte giving the length of the <data> portion, in bytes. Data length is between 0 and 128 bytes.

<data> -- <data length> bytes of data. This data can be either ASCII characters or binary bytes.

<crc> -- Two bytes of cyclic redundancy check. The CRC is a type of checksum, and it covers everything from <cmd> to the end of <data>, inclusive.

In order to assure that <10h><03h>, the beginning of frame marker, does not get transmitted in the frame, all <10h> bytes other than the one at the beginning of a frame are doubled. That is, during transmission, <10h> is converted to <10h><10h>. When receiving a frame, after the first <10h><03h> has been detected, all <10h><10h> sequences should be converted to a single <10h>. If a non-doubled <10h> is encountered in a frame, it is an error.

2.3 MSG2 CRC

Every frame transmitted by the MSG2 ends with a two-byte Cyclic Redundancy Check (CRC). The CRC is an error detection code, and if you use the CRC equation on a received frame, your two-byte answer should match the two bytes transmitted at the end of the frame. The CRC used by MSG2 is calculated using a modified CCITT CRC algorithm. A Z80 machine-language program showing how this is done is provided in the appendix.

The CRC calculation includes all bytes from <cmd> to the end of <data>. The CRC calculation is done prior to doubling <10h> bytes and, by the receiver, after removing the extra <10h>.

2.4 Title Frames

The DCE was required to "do something interesting" when it was idle, i.e. not performing a function at the specific request of a ground station. To this end, MSG2 sends the first line of each active message on the downlink when it is idle. This line is the message "title" and usually contains at least the source, destination and subject of the message. Ground stations can see if they have any waiting traffic without interacting with the DCE by simply copying these title blocks. The OBC DIARY program currently switches the DCE onto the downlink for 30 seconds at roughly 5 minute intervals.

Title frames provide a way for stations not directly involved in DCE operations to monitor DCE activity. The <cmd> byte in a title frame is "T". The contents of the <data> portion of a title frame are are as follows:

Message number, 1 byte. If the first bit of this byte is set, the message is not complete, and the message title may be invalid. Message numbers for complete messages run from 0 to 127.

Message length, 1 byte. This is the length of the message that is stored on the DCE, it is not the length of this title frame. Multiply this by 64 to get the message length in bytes.

Call sign of station using DCE, 9 bytes of ASCII. If no one is using the DCE then this will be 9 blanks.

Title of the message, the remaining <length> minus 11 bytes of the <data> field. This is taken from the first line of the message. The length referred to above is the FRAME LENGTH (which follows the inverted command). The 11 accounts for the message number, message length and call sign data.

The title for message number 0 contains MSG2 administrative and status information. It currently contains MSG2 version number, a counter from the error detection and correction (EDAC) memory, the number of free memory blocks available, the number that will be assigned to the next message, a counter that is incremented every time MSG2 receives a valid frame, an error indicator and an indication of which bank of RAM is active. Message 0 itself is used to download portions of the program variables, including a table of memory address where the EDAC circuits have corrected an error.

2.5 Other Frames

The above information and a short computer program will allow causal ground observes to monitor DCE activity. During actual DCE operations, however, several other frame types are used. The following command frames are used by DCE ground stations, and the list provides insight into the operation of the MSG2 mailbox.

LOGIN tells the DCE the call sign of the ground station.

LOGOUT frees the DCE for use by another ground station. Logout is automatic if the DCE does not hear the ground station for two minutes.

PUT is used by the ground station to store a message to the DCE.

CONTINUE allows the ground station to continue (on another orbit) a PUT operation that was interrupted by LOS.

GET is used to retrieve a message from the DCE.

KILL deletes a message.

END resets DCE software to the title-display mode, without logging out the ground station.

Thus, the DCE has all of the commands needed in a computer bulletin-board system.

3.0 DCE SOFTWARE

The DCE MSG2 software is implemented in 280 assembler code. It is in assembler for both size and speed, the DCE MSG2 runs in 2.5k and supports full duplex operations at 1200 baud on a 280 with a .9MHz clock.

The program resides in the memory protected by hardware EDAC. Messages are stored in 96k of non-protected memory. This memory is mapped into the upper 32k of the 280 memory space. There are two 32k banks and two 16k banks. The banked memory is organiied as a linked list of 256-byte blocks. All of the banked memory is used except for the block in bank three containing location A5F1h, which has a bit that went bad shortly before launch. Messages consist of a linked list of memory blocks, unused blocks are kept on a free list.

All of the link pointers for the memory blocks are kept in the memory protected by hardware EDAC. Currently, blocks from deleted messages are returned to the top of the free list where they will be the next to be re-allocated. This tends to kept bank 1 in use and bank 4 empty. This will be changed in a future version.

The banked memory is not currently protected against charged-particle induced soft errors. Algorithms are being developed to provide this memory with software EDAC in the future. In 60 days of monitoring, we have only seen three errors in the 16k of hardware-protected memory. It is hard to draw conclusions from this limited data, and the memory technology is not the same in the banked memory. Experiments are planned to gather data on soft errors in all parts of memory.

3.1 Other MSG2 Functions

The MSG2 software on the DCE supports two non-message related functions.

BYPASS - The DCE sends all characters received through the VHF uplink to the UART that leads to the spacecraft command system. This provides backup to the similar function performed by the DIARY program on the 1802 computer in case of 1802 failure or the need to completely reload 1802 software. The BYPASS software resides in the receive interrupt handler, and therefore should perform the bypass function with high reliability no matter what the other levels of MSG2 software are up to. There is no way to disable the BYPASS. The BYPASS was made necessary by a

failure in the VHF uplink facility shortly after launch.

MEMWASH – This function washes any accumulated single-bit errors out of the hardware EDAC memory by reading a location and writing it back. A different memory byte is fetched and stored once on each trip through the main loop of the software, which occurs at least once every 9.2ms. All of the EDAC memory is checked at least once every 150 seconds, this number might be 10 to 100 times faster, depending on uplink activity.

A third function under development is the gathering of information on soft memory errors in the unprotected areas of memory. This is to further the DCE's role in gathering engineering data used for PACSAT development as well as to provide a high level of data integrity for messages stored onboard the DCE.

4.0 MSG2 GROUND STATION SOFTWARE

The MSG2 protocol provides a small number of basic features: logon, logoff, put, get, and kill. The ground station software supplies additional "user friendly" facilities. Such features include remembering the start point for partial messages, providing wildcard or multiple file transfer, automatic logging, and providing antenna pointing cues; these features are not part of the basic set of functions, but make the DCE easier to use. The features available at any particular ground station depend on that station's hardware.

5.0 MESSAGE FORMATS

MSG2 is a data-transparent system, i.e. messages are stored as a single string of 8 bit bytes. Message content does not effect and is not effected by communication through MSG2. Most messages, however, will follow a fixed format for their first line. The first line is defined as the text up to the first <cr>, or 116 characters. This is the part of the message that is sent on the downlink in title blocks.

5.1 Person-to-Person Messages

The following format is used for standard messages:

To:<call> De:<call> Re:<title>

The call can be up to 9 characters. There are no spaces after the colon in any field.

For example:

To:GO/K8KA De:NK6K Re:Software updates

The To: and De: fields are the call signs of DCE ground stations. A future command in MSG2 will permit messages in this format to be searched by To: field and downlinked in a group. The format is flexible, and fields may be added to it if the DCE is used for other than direct ground station to ground station data transfer.

6.0 THE FUTURE OF THE DCE

Several hundred kbytes of data have traveled between UoSAT headquarters in Surrey (UK) and NK6K in Los Angeles (USA) on the DCE. MSG2 ground station and satellite software works efficiently and reliably. While much of the future use of DCE store-and-forward capability depends on radio-regulatory matters beyond our control, the DCE has successfully proven that store-and-forward communications using a satellite in low-earth orbit can be carried out routinely. It has put us in a position to make informed design decisions while working an a proposed dedicated store-and-forward spacecraft.

The limited memory available on UO-11 and the fact that DCE activities consume bandwidth on the UoSAT-11 command uplink and general downlink dictate that only a limited number of selected ground stations will take part in future DCE communications. These ground stations will be chosen such that, regulations permitting, each can serve as a gateway -- delivering amateur radio news and and technical information to stations outside of the DCE ground station network.

Equipment for an East Coast North America gateway is in place and should be operational by the time this sees print, A station will be brought on the air soon in New Zealand. Discussions are under way for stations in Australia and Japan. The DCE is ready now to serve as an effective link between the amateur radio service's far flung packet radio networks.

7.0 SUMMARY

The DCE project was begun as an opportunity to gain experience in the design, construction, and orbital operation of space-based large-memory store-and-forward message relay satellites. The parts used in its construction are giving us data on the suitability of high density non-specialized memory devices and microprocessors (i.e. inexpensive) in low earth orbit, which is directly applicable to future amateur space projects. The coming months will give us experience scheduling gateway operations to maximize data volume in a worldwide network of ground stations.

The capability exists now to move 114k bytes per day to or from a single ground station.

The experience gained from writing MSG2

software, managing DCE operations, and dealing with regulatory issues will be of great use when the mailbox on JAS-1 becomes operational and when design teams set to work on PACSAT -- an amateur radio satellite dedicated to store-and-forward communications.

Appendix 1 - Acknowledgments

Here is a list of the North American crew who worked on the design and construction of the DCE, in no particular order:

Stan Kazmiruk, VE3JBA; Dick Atkinson, VE3JBO; Gord Scale, Bob Gillies, VE3JA; Hugh Pett, VE3FLL; Geoff Clarke, VE3JBD; Dale Ward, George Roach, VE3BNO; Grant Bechthold, VE3JBF; John Henry, VE2VQ; Ron Archer, VE3CNM; Murray Gold, VE3KHG; Larry Kayser, WA3ZIA; Dave Cheek, WA5MWD; Chuck Green, NOADI; Lyle Johnson, WA7GXD; Harold Price, NK6K; Bill Reed, WDOETZ; Jose Sancho, WB5YFU; Bob Stricklin, N5BRG. Richard MacBeth, G8VLY, assisted in the final close-out at UoS. Funding for DCE hardware was provided by Volunteers in Technical Assistance as part of their support of the PACSAT project. Gary Garriott, WA9FMQ is the VITA interface. Acknowledgment is given to AMSAT for its continued support, and to the UoS team.

Additional information on the DCE hardware, design, and construction can be found in a paper by Lyle Johnson, WA7GXD, "The OSCAR 11 Packet Experiment", Proceedings of the Third ARRL Amateur Radio Computer Networking Conference.

Appendix 2 - DCE CRC algorithm.

The routine below can be used to compute the checksum for reception of DCE frames. The HL register is cleared before the first byte is received. Each byte is acted on in turn. When all bytes have been checksummed, the result is compared against the received checksum. The L register contains the first byte received, the H register the second.

```
;
; COMPUTE CRC ON A, INTO HL
;
CKSUM:
        LD      B,8
        LD      C,A
CRC2:
        LD      A,C
        RLCA
        LD      C,A
        LD      A,L
        RLA
        LD      L,A
        LD      A,H
        RLA
        LD      H,A
        JR      NC,CRC4
        LD      A,H
        XOR     10H
        LD      H,A
        LD      A,L
        XOR     21H
        LD      L,A
CRC4:
        DEC     B
        JR      NZ,CRC2
        RET
```

In using this program on DCE frames, remember that the CRC covers all bytes from the <cmd> to the end of the <data> segment, inclusive,' It does not include the CRC itself, or the leading <10h><03h> bytes. Also, CRC calculation is done prior to doubling <10h> bytes and, by the receiver, after removing the extra <10h>. To check your CRC program, CRC check the characters "TEST MESSAGE". The result should be CRC bytes L=253 and H=223. 253 would be the byte transmitted or received first.

AUTOMATED TRAFFIC HANDLING ASSISTANCE

David Cheek, WA5MWD
1510 Travis Street
Garland, TX 75042

Packet radio presents an opportunity to improve speed and accuracy of message handling. Speed is often limited by the typing speed of the operators. The accuracy is assured during transmission, but is only useful if the message is correctly entered into the message handling system. The normal limits to message handling include a lack of fully qualified operators, and inability to use untrained people during special situations such as disaster events. A method I have used to improve both of these is keystroke reduction.

Fill in the blank programs are useful to assist newcomers in preparing traffic. These help if they keep inaccurate messages from entering the system, and if they allow professional typists to do high speed preparation of traffic off line. This traffic would be handled by a smaller number of amateurs operating the radios. This method has been used without keystroke reduction. With it, a smaller number of off line operators is needed to keep the traffic circuit busy. The radio operators should only spend about ten seconds per message sending and confirming receipt of "prepared messages." The normal pace of message handling by a good CW operator is one per minute. We should be designing and testing for a six fold increase in speed.

The most basic method of automating this process is to save fields that do not change over several messages. For a large operation this includes, precedence, handling instructions, station and place of origin, and date. In some cases, much of the text may be the same also. These can either be filled in and transmitted clear test, or "booked" and sent once at the start of each book.

Automatic sequence numbering is normally tried next. This works fine if only one computer is used for text entry. If two or more are used for off line for text entry, then a method to prevent duplication of message numbers must be included. I used a limiter, which allow each operator to start at a specific message umber, assigned by a single coordinator, and only enter ten messages. In this way, several operators will not use the same number over. I expect ten messages to take five minutes for typing.

The best timesaver of all is automatic word counting. I implemented this in both basic and in 8080 assembly language. It counted letter groups separated by spaces. This seems to match the rules used by the ARRL in counting words. One special case had to be handled, the last word at the end of a line. The assembly language version had to take special care to treat non printing characters properly. Delete characters did not break words, and neither did backspaces. Carriage returns, however, were similar to spaces and I included both as "word delimiters." The basic version counted about 20 characters a second, using an interpreted version of BASIC. The assembly language version has been to fast to bother with timing. .A further benefit of this module is that is the text can be located in the received message, it can be counted and compared with the "check" field, further saving time for the receiving operator.

None of this is any value unless the implementation allows a typist to back up and correct text in error. One version, from N5EZM, allowed the preamble to be reviewed and changed when first entered. No further correction was provided. after the last field was complete. If anything was wrong, the message had to be fixed with a text editor. This slows down the whole process, and discourages the correction of small errors. Error correction must be included in the message creation package.

Tests were not useful because only one computer was used for both text entry and on air operation. Also, we have not had an event present enough traffic to reach the planned rate of message handling, over sixty messages per hour.

The PACGRAM application allows all parts of an ARRL radiogram to be identified by a computer by delimiting each field.

This allows routing of messages by city/state, and automatic word counting at receive points to check for flow control problems. While PACGRAM creates long frames, operation with PACLEN set to smaller values can be used to improve operation on marginal paths. The PACGRAM definition is included as an attachment.

Automatic message assistance applications have potential. They will be most useful where large volumes of similar traffic are generated, or when fast liaison with the NTS is required. Official traffic for disaster relief agencies is not likely to benefit as much from these developments.

> > > > > > > > > > Pacgram Protocol Definition < < < < < < < < < <
(Versions 1.5.0 and later)

By: Jay Nugent, WB8TKL
3081 Braeburn Circle
Ann Arbor, Michigan 48104
Dated: 860128

PACGRAM is an application software package that runs on the host computer connected to a TNC. The PACGRAM software is responsible for prompting the operator for the proper Radiogram information one field at a time and forms a PACGRAM message from this. The message can then be sent to the TNC for transmission into the network.

On the receiving end, PACGRAM decodes the data stream from the TNC for the starting characters of a PACGRAM. When it finds these characters it receives the rest of the message and can later decode it back into the Radiogram format for display on the console, or as printed on the printer. Received PACGRAMS are stored in buffer space and/or disk files and maybe later retransmitted for forwarded to other stations in the network.

Special characters are used within the PACGRAM to signal the start of the PACGRAM message, the end of the PACGRAM message, and to separate the fields of information within the PACGRAM message.

These control characters and the protocol are described in the following definition.

--> PACGRAM CONTROL CHARACTER and PROTOCOL DEFINITION

The control characters, and character sequences, used in PACGRAM were based on the unlikelihood that they would ever appear in part of a normal Radiogram. Consideration of the CW traffic nets that will handle message traffic generated with PACGRAM was also taken into account since many characters cannot be sent using CW.

For those stations not possessing PACGRAM software, these control characters were selected so that a PACGRAM can be read directly from a terminal and written back into the standard Radiogram form very easily by hand. A Formsmode has been added to the protocol to allow the sending of a directly printable PACGRAM. This enables any station to receive a PACGRAM already formatted to be printed on hardcopy. This form of PACGRAM uses its own starting sequence that can be easily detected by a small computer running BASIC. Once the start sequence is detected, it can then route all output to the printer. The standard end of PACGRAM character is used to indicate the end of the print so that the output can then be routed back to the console.

==========

--> The START of a PACGRAM shall be a pound sign '#' followed by asterisk '*'.

The purpose of this character sequence is to signal the start of a PACGRAM and differentiate it from any other data sent by the TNC to the host computer. Such as other communications data or commands and responses from the TNC.

The start of the PACGRAM sequence has been altered since the first release of this protocol in an effort to avoid false starts caused by WORLI like PBBS's that use the # character in their data.

--> The DELINEATION character shall be an asterisk '*'.

This character is present in the PACGRAM to delineate the standard Radiogram fields from one another. The absence of data in any one of the fields will cause two consecutive asterisks to appear within the PACGRAM. No filler characters are placed between the asterisks of an empty field.

--> The FIELD ORDER of a PACGRM is as follows.

NUMBER / PRECEDENCE / HANDLING INSTRUCTIONS / STATION OF ORIGIN
CHECK / PLACE OF ORIGIN / TIME FILED / DATE FILED

NAME TO / NUMBER & STREET / CITY / STATE / ZIP / PHONE NUMBER

TEXT OF THE MESSAGE

SIGNATURE / TITLE OF SIGNEE

Note: The CITY and STATE fields have been separated into two individual fields in this version of the protocol. This allows for automated routing based on the destination City and State.

--> FIELD LENGTHS and TYPES within the PACGRAM.

The Number field will be limited to 8 characters maximum.
The Check field will be limited to 2 characters maximum.
(This may be expanded to handle ARL type checks)
The Text field is limited to a maximum of 1024 characters.
All other fields are limited to 60 characters.

Field types may be either alphabetic or numeric but characters that cannot be sent using CW will not be allowed.

--> The END or a PACGRAM shall be the ampre sign '&'.

The purpose of this character is to signal the end of the PACGRAM.

--> The START of a FORMSMODE PACGRAM shall be '#PAC&' followed by a carriage return.

The purpose of this starting sequence (including the carriage return) is to signal the start of a PACGRAM in the FORMSMODE. A small computer running a BASIC program can trap this starting sequence and then direct all its output to a printer.

--> The END of a FORMSMODE PACGRAM shall be an ampre sign '&' followed by a carriage return.

The purpose of the end of FORMSMODE sequence is to signal the end of a FORMSMODE PACGRAM. A small computer running a BASIC program can trap this sequence and then direct its output back to the console.

==========

As a Radiogram has well defined fields in a specific order, so does the PACGRAM. The order in which the fields occur in the Radiogram is the exact order that they will appear in the PACGRAM. For example, here is a sample Radiogram followed by its equivalent data stream in PACGRAM form.

NUMBER: 126 ROUTINE WB8TKL CHECK: 5 ANN ARBOR MI 14302 MAY 21
TO: MIKE NUGENT
123 HOLLYWOOD AVE
HOLLYWOOD, CAL 54321
(818)555-1234
TEXT: HOW IS THE WEATHER X
SIGNED: JAY

and the equivalent in PACGRAM form would be:

#*126*R**WB8TKL*5*ANN ARBOR MI*1430Z*0521*MIKE NUGENT*123 HOLLYWOOD AVE*HOLLYWOOD*CAL*54321*(818)555-1234*HOW IS THE WEATHER X*JAY**&

As you can see, the length is well within 256 bytes, the maximum AX.25 packet length. Even with the Paclen set to 256, a single packet could contain a fairly large text field. You can see the benefits of using PACGRAMs over the voice or CW methods of sending traffic. This packet can be sent in just a fraction over one second at 12 bps, an enormous improvement over existing amateur traffic systems.

Also notice that in my example, since I left the fields for Handling Instructions and Title blank, that PACGRAM simply put no data between the two asterisks. This is necessary to maintain the field count for decoding of the PACGRAM at the receiving end and also wastes as little of the transmission bandwidth as possible.

Happy Packeting -WB8TKL

PACKET RADIO DEMONSTRATIONS AS A SUPPLEMENT TO CLASSROOM INSTRUCTION1 IN TELECOMMUNICATIONS

by

Robert J. Diersing, N5AHD
Associate Professor of Computer Science
Corpus Christi State University
6300 Ocean Drive
Corpus Christi, Texas 78412

February, 1986

Abstract

During the past year the author has had the opportunity to use packet radio hardware and operations to demonstrate concepts taught in telecommunications courses at an upper-level university. This article provides a brief discussion of how this was accomplished. A description of the courses and their intended audience has been included.

Introduction

Corpus Christi State University is a state-supported upper-level institution with an enrollment of approximately 3,700 students. There are approximately 300 students majoring in computer science. About 100 are graduate students, and the remainder are seeking the bachelor% degree. Both the undergraduate and graduate curricula are oriented more toward applications of computing rather than theoretical computer science.

During the past year two different courses were offered. An undergraduate course, CS 442 Teleprocessing, deals primarily with terminology and various telecommunications software systems used on IBM mainframes. Data Communications Software Design by Malcom G. Lane is the textbook used in the course. A graduate course, CS 555 Telecommunications Systems, deals with more theoretical issues and in particular the networking of computer systems. The textbook is Computer Networks by Andrew S. Tanenbaum. Tanenbaum's book needs no introduction. Lane's text, which is based on the work of Tanenbaum and others, includes chapters network architectures, data link control protocols, and protocol definition.

It is important to note that unless the student has had experience with computer networking through his/her job, dialing into a local TELENET, TYMNET, or UNINET port may be as close as any have been to networking. None of the university's computers are interconnected,

Presentation of Packet Radio Hardware

The end of Tanenbaum Chapter 4 is an appropriate time for the first presentation of packet radio hardware. The few components in a Terminal Node Controller can be easily presented in a block-diagram approach. Thus, the student gets some concrete idea of how textbook examples might be realized. The fact that the physical level is implemented in hardware takes on new meaning. Data link layer functions such as framing can also be tied in particularly if time is taken to review the functions provided by most HDLC controllers. A parallel can be drawn between the functions performed by a TNC and a packet assembler/dissassembler during discussions of packet switched public data networks.

Now the function of hardware and software to be governed by the X.3, x.28, and x.29 standards can be more easily understood.

For actual on-the-air demonstrations approximately six stations were available. One TNC and radio was in the classroom. There was another system in another room in the same building. A station was on the air at the author's residence and a few other stations were on in the city.

Presentation of Packet Radio Protocols and Operations

While packet radio hardware in the form of a TNC reinforces certain concepts, the operational aspects of a packet radio network provide even more examples to solidify theory. When presenting the AX.25 protocol it is important to note that it was designed for a half-duplex broadcast medium as opposed to a full-duplex point-to-point link. A demonstration of such a network even in a relatively low traffic area such as Corpus Christi can be extremely beneficial.

It should also be noted at this point that previously published papers on amateur packet radio protocol and network development provide a large and pertinent collection of supplemental material. Most notable are the AX.25 Amateur Packet-Radio Link-Level Protocol Standard published by ARRL and the AMICON System Specification by H.S. Magnuski, KA6M.

Even, with only a few stations available, the problems of collisions and congestion in broadcast networks are apparent. The demonstration of digipeaters clearly shows the impact of end-to-end acknowledgements. The 0.19 theoretical maximum efficiency of a pure-broadcast network becomes entirely believable. Address assignment issues can be presented. Of course all these issues can be decided on a purely theoretical basis, but the demonstrations supplement the textbook material very well.

The operational demonstrations were given after Tanenbaum Chapter 6, Satellite and Packet Radio Networks. A discussion of the UoSAT-2 DCE, PACSAT, and JAS-1 spacecraft was included at this point.

Packet Radio and Satellites

With less than a month remaining in the graduate course last fall, our student programming team left for the ACM South Central Regional Programming Contest. Two students from the Telecommunications class were on the team. Imagine their surprise when the first problem statement began, "A text message has been received from the ACMSTAT satellite. The message is encoded as a file of packets named PACKETS.DAT. Each packet contains seven characters along with error-detection and correction information..." The team captain later reported many other teams remarking to themselves, "What is this... packets and satellites?". Did the demonstrations help? Only four of the forty-three teams completed the problem. The exposure to packet radio applications on earth and in space couldn't have hurt.

Conclusion

It has always been a personal goal of the mine to bring theory and application together. It is a rare opportunity when a hobby and a profession can be synthesized to accomplish that goal. From a teacher's point of view the important outcome was a few more concepts obviously more

clearly understood. From a radio amateur's point of view the important outcome was one licensed amateur and two more studying for the exam. Clearly it was a productive semester.

Acknowledgements

I would like to thank Phil Karn, KA9Q, who provided important comments on course contents. I should also point out the EIES "protocol wars" provided more than a few lecture notes. My thanks to all.

Trademarks

Telenet, Tymenet, and Uninet are registered trademarks used in this paper.

OUTLINE OF SATELLITE JAS-1

JS1UKR, Fujio Yamashita
Japan Amateur Radio League

Introduction

The first Japanese satellite, JAS-1, is scheduled to be launched in 1986 by Japanese H-1 rocket. A special feature of JAS-1 is its digital transponder with memory, in addition to a normal analog transponder. It will be possible to upload digital messages into the transponder memory, and the messages will be relayed to someone with the appropriate access code (e.g. callsign). In this way, JAS-1 will be able to carry messages (on a store and forward rather than real-time basis) between amateurs anywhere in the world.

Two birds of JAS-1 were completed in the fall of 1985 and all necessary testing was carried out and certified that the characteristics of satellite were no problem. The cost for constructing this satellite is about 400 million yen (US$ 1.6 million).

Major Specification of JAS-1

Launch and Orbit

Launch at	:	early in August, 1986
Launch by	:	NASDA, with H-1 rocket
Launch from	:	Tanegashima Space Centre, Japan
Orbit	:	circular, altitude of 1500 km
Period	:	120 minutes
Inclination	:	50 degree
Life	:	3 years

Construction

Weight	:	50 kg
Configuration	:	polyhedron of 26 faces covered in solar cells
Size	:	400mm(dia.) x 470mm (height)

Communication Subsystem

Analog (JA) and digital (JD) communication in mode J.

Transponders

Analog Transponder (linear transponder)

Input frequency: 145.9 - 146 MHz (bandwidth 100kHz)
Output frequency: 435.9 - 435.8MHz (inverted sideband)

Reqd. uplink eirp: 100 W
Transponder eirp : 2 W PEP

Digital Transponder

Input frequency: 4 channels, 145.85, 145.87, 145.89, 145.91 MHz
Output frequency: 435.91 MHz (1 ch)
Reqd. uplink eirp: 100 W
Transponder eirp : 1 W rms
Signal format : 1200 baud PSK, store and forward

Beacon and Telemetry

JA beacon: 435.795 MHz, 100 mW, CW or PSK
JD telemetry: 435.910 MHz, 1 W, PSK

Figures 1 and 2 show the structure and the system block diagram of JAS-1.

Consisting units of communication subsystem are named as follows:

JRX	:	receiver of both analog and digital signals
JTA	:	transmitter for analog signal
JTD	:	transmitter for digital signal
DDEM	:	demodulator for digital signal
DCM-A	:	digital communication module A
DCM-B	:	dit. B

Digital Transponder

Digital transponder consists of JRX, DDEM, DCM-A and B, and JTD. Figure 3 shows the block diagram of communication subsystem. Uplink signals of four chan-

nels are accepted at the same time, while downlink signals are delivered only through one channel, for efficient use of traffic. Frequency spectrum for JAS-1 is shown in Figure 4. Digital QSO is possible not only for station to station, but for BBS or sending telemetry data of satellite in packet, by means of the protocal 25, level 2, version 2.

The uplink signal carries Manchester-coded NRZI/HDLC data, and this signal is demodulated by DDEM, that contains four communication channels and one command channel. Output signals of DDEM are processed by four-channel HDLC of DCM-B to deliver them to CPU of DCM-A. DCM-A is composed of CPU, NSC-800, memory of 1 Mbyte made of 256 kbit DRAM, A-D converter, encoder and so on. CPU processes signals of communication and telemetry signals, and analyses command signals. Communication signals, as well as telemetry data, are transformed into packet by HDLC of DCM-B, and JTD transmits these signals by PSK modulation, at the frequency of 435.91 MHz, 1 watt of power.

Antenna

Antenna system consists of three parts: the receiving antenna, the transmitting antennas for both analog and digital signal, all having almost omni-directional characteristics. The receiving antenna is a monopole set at the top of the satellite. Each of two transmitting antennas has four elements around the satellite. These are connected to the antenna power divider (APD) and phase shifting cables, to make the radiation pattern circular. Radiation is LHCP looking the satellite at its bottom.

QSO via Digital Transponder

Equipments for digital QSO via JAS-1 are not so different from that on the ground link, except modem for PSK signal.. As the communication control device, TNC of AX. 25, level 2 is available. Current communication equipment may not have PSK modem, so it is requested to prepare PSK modem. Transmitter for uplink should be 145 MHz of FM and downlink receiver, 435 MHz of SSB.

Bit rate of data is 1200 bps and the modulated signal for uplink is F2 and for downlink, SSB. Data to be transmitted is transformed into Manchester code, by clock signal, produce F2-modulated RF signal of 3 kHz bandwidth. Downlink signal is transmitted by SSB with PSK modulation, and this is converted into audio frequency, and is synchronously detected. After detection, data are fed to TNC, through a low-pass filter and adjusting amplifier. An example of the modem for communication via JAS-1 is shown in Figure 5. It is important and interesting problem to overcome Doppler effect in various passes of the satellite.

Closing Remarks

Project of JAS-1 was planned and has been promoted by JARL, under which the amateur satellite committee has taken the role of steering the plan. Manufacturing of the satellite is carried out by NEC Corporation of Japan and the satellite repeater group of the committee. The satellite repeater group, of which members are excellent staffs from the JAMSAT (Japan AMSAT), made both analog and digital transponders and the digital memory by their own hands. NEC is responsible not only for manufacturing of power system and satellite structure but also for system design and integration. This work is supported by many people in the world, and here we are to thank them heartily. And we are expecting nice flight of JAS-1 and nice QSOs all over the world via JAS-1.

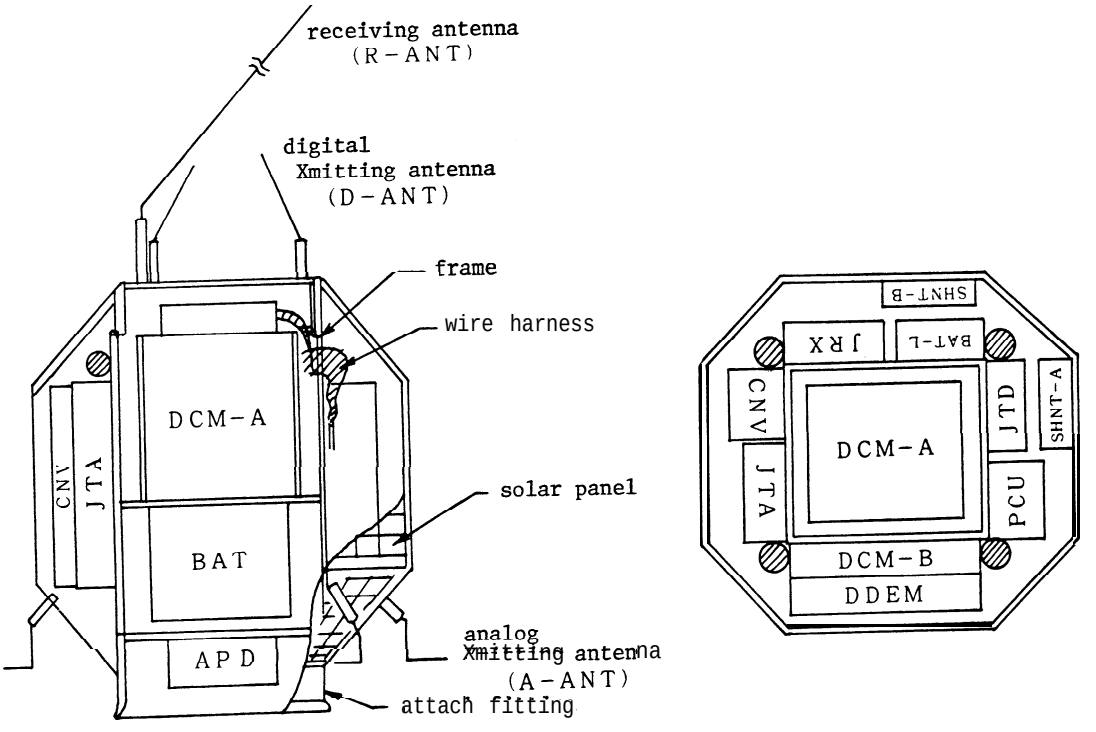


Figure 1. Structure of JAS-1

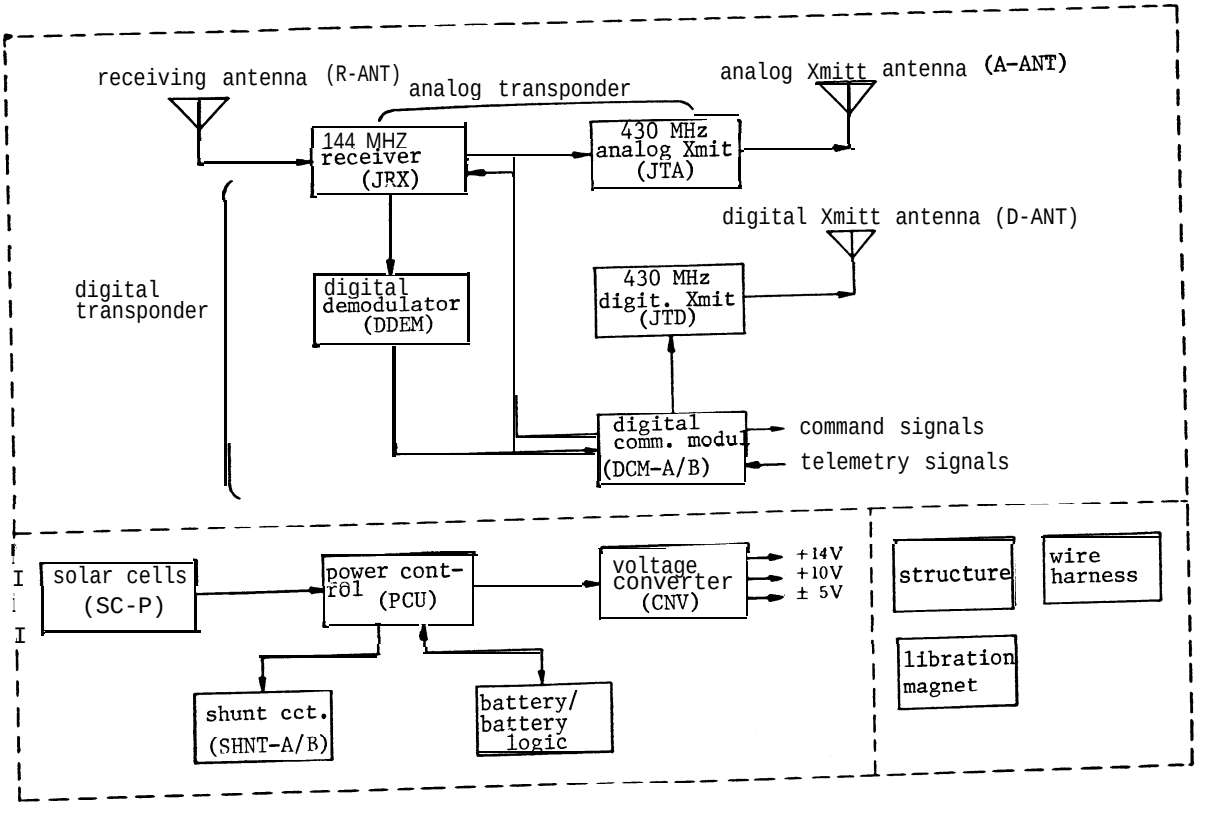


Figure 2. Block diagram of JAS-1 system

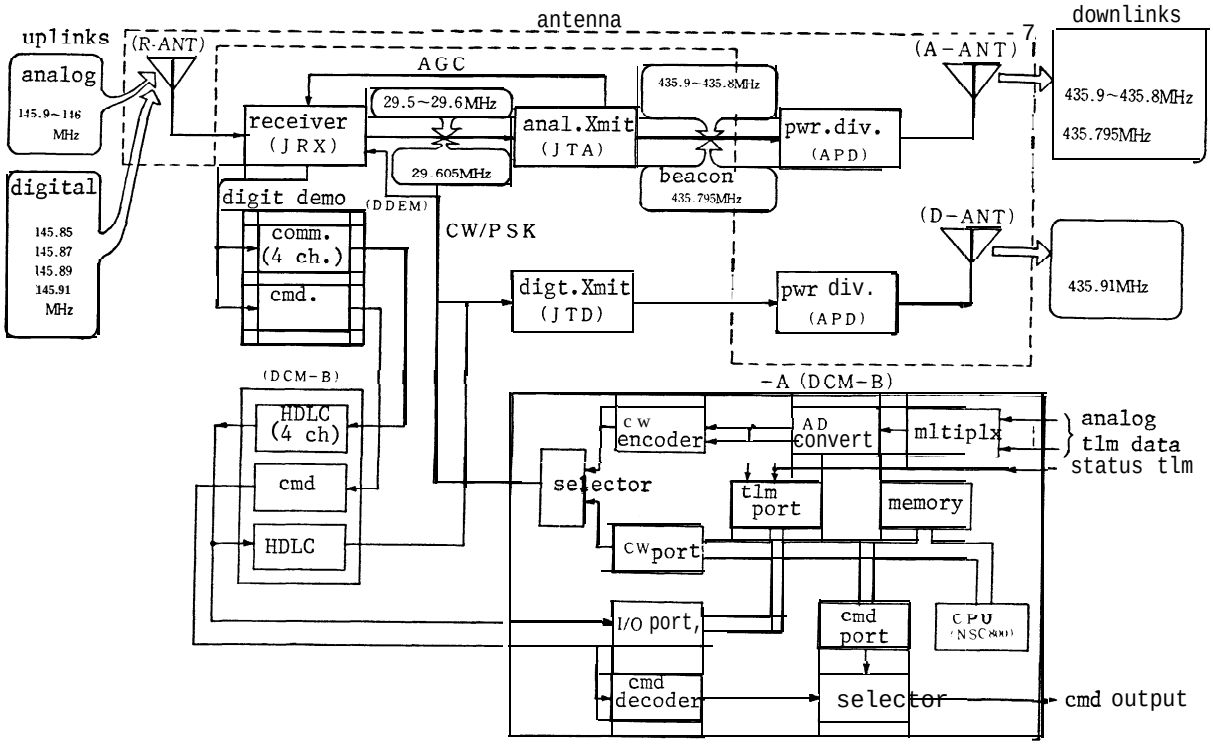


Figure 3. Block diagram of communication subsystem

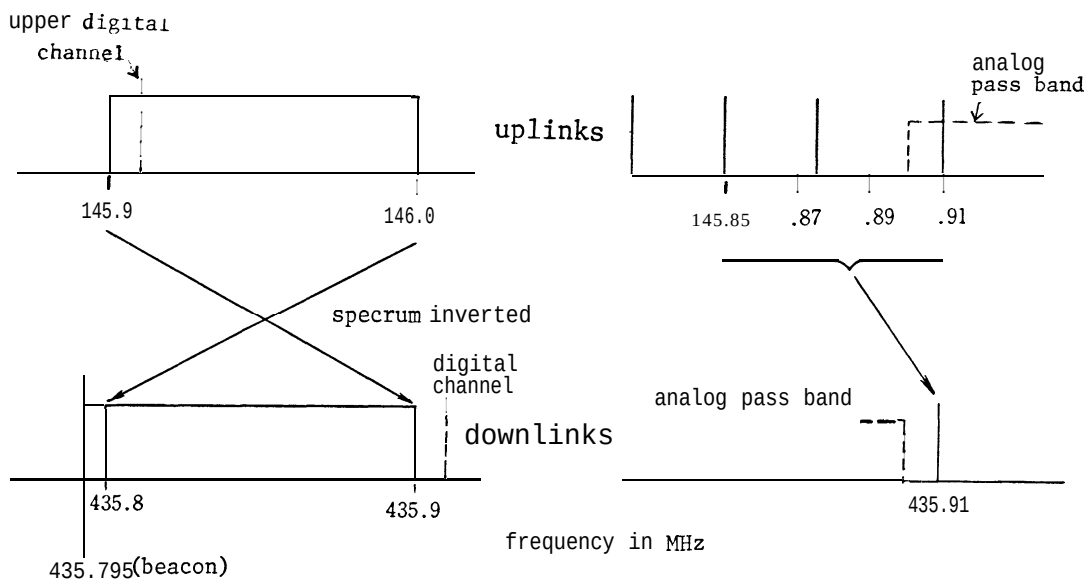


Figure 4. Frequency spectrum of JAS-1

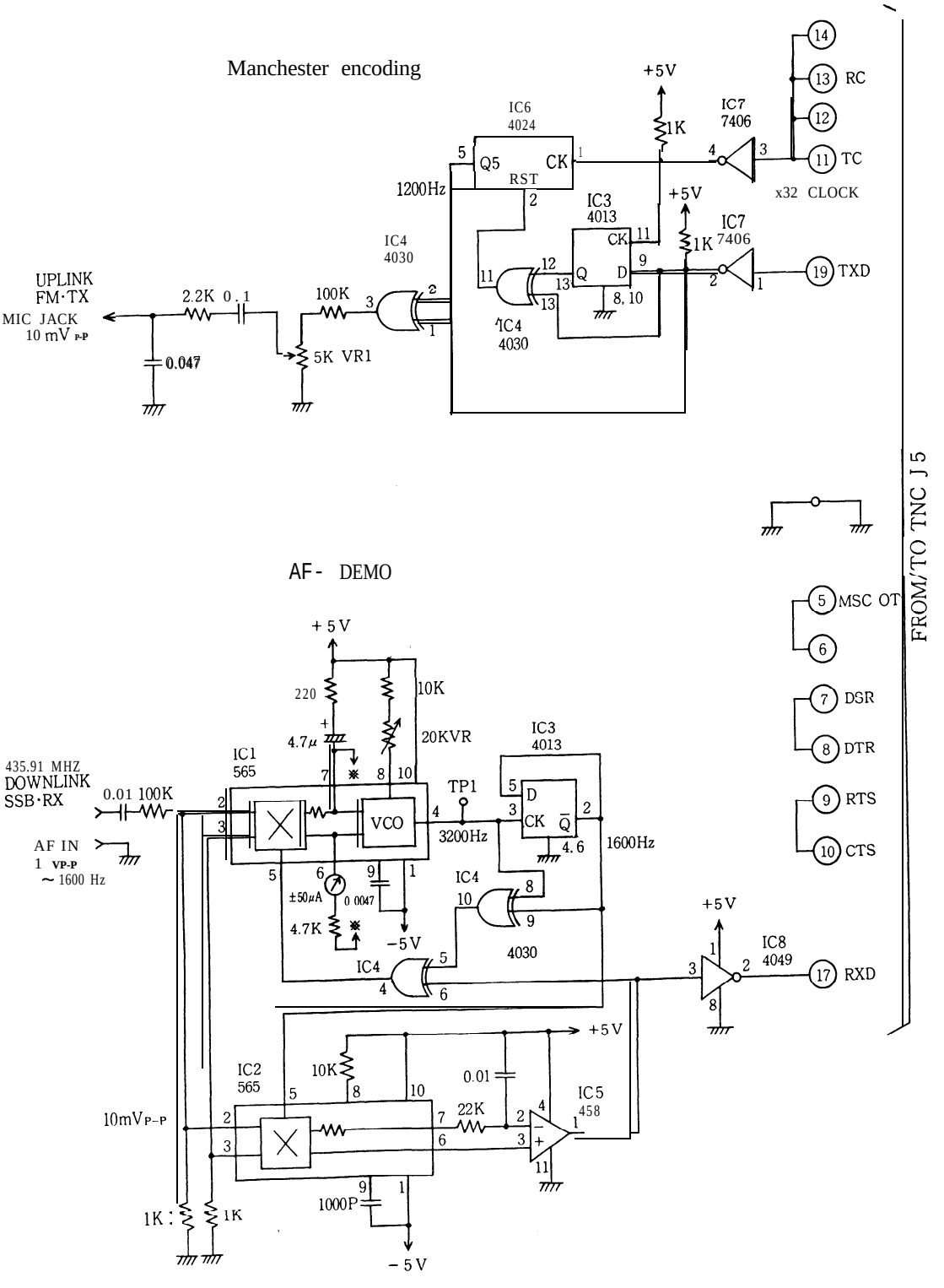


Figure 5. An example of modem circuit

What's All This Racket About Packet?

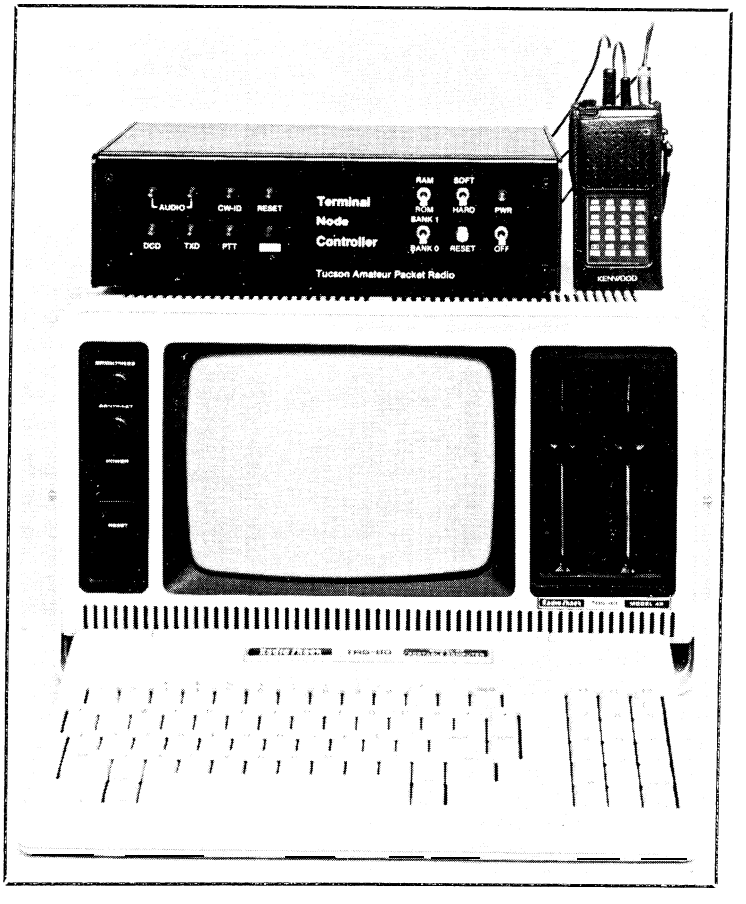
Packet radio is growing *fast.* What's it like? Read on.

By Harold Price,* NK6K

This is an article about packet radio. If you don't know what packet radio is, that pronouncement won't mean much to you. "High Speed CD/CSMA Digital Communications in the Amateur Radio Service, Theory and Applications" isn't a real grabber either, so "this is an article about packet radio" will have to do. This article will have a slant different from most previous articles about packet radio. It will not discuss what you will be able to do with packet in the future, but what you can do with packet now: how to use the existing networks, which of several frequencies in your area are being used for packet, what packet controllers are available and what you can expect when you get on packet radio.

After we discuss what packet does, there will be some theory-an explanation of how packet does what it does (there's no such thing as a free lunch). Those of you who want to dig deep into technical topics should consult the bibliography at the end of this article.

A final word of warning: This is a sales pitch! The goal is to get you interested enough in packet radio that you will get involved. Whether you visit a friend's packet-equipped shack, see packet in action. at a radio store or Field Day site, go to local packet-radio meetings, or jump in and buy or build a packet controller, you'll learn far more by doing than by reading.



Executive Summary

Packet radio, or simply "packet," is the common name for a digital communications mode in Amateur Radio that provides error-free communications. It is designed to allow automatic linking of systems for cross-country networks. Packet uses high speeds; 120 characters per second is standard on the 2-meter band, and a recent development permits 960 characters per second on the 220-MHz band. Below 30 MHz, 30 characters per second is used. Assuming that you already have a radio and a computer or terminal, it will cost you between $180 and $500 (higher cost for more "bells and whistles") to get on packet radio. This is the cost of a *terminal node controller* (TNC), a box that connects the data-generating device to the radio. In other words, packet allows hams to exchange information much faster than before with no errors at a reasonable cost.

What Is Packet?

Packet is usually character communication. Letters and numbers entered on a keyboard or from a computer are sent from one amateur station to another. On the surface, this sounds no different from RTTY, which has been around in Amateur Radio for many years. Packet radio has three major characteristics and several beneficial side effects that make it stand out from other amateur digital communications modes.

Packet radio guarantees perfect reception.

Information sent via packet is checked to see that it was received exactly as it was transmitted. Data is automatically retransmitted until it is accurately received. There is only a very small chance (one in millions) that bad data will sneak through.

Packet permits a single frequency to be shared by several simultaneous conversations.

This also makes possible a bit of magic called a "simplex repeater"-a repeater with its input on the same frequency as its output.

Packet allows for "routing" information among stations, so that any packet station can be part of a linked set of "repeaters."

* 1211 Ford Ave., Redondo Beach, CA 90278

For example, there is a set of packet stations in California that allows information to be sent from San Diego to Sacramento, a distance of 480 miles, on 145.01 MHz. Before users of mega-linked VHF/UHF FM repeater systems start chuckling at this "paltry" distance, consider that each station on the packet link uses a single frequency, a single antenna and a single transceiver. Stations do not require additional telemetry, diplexers, duplexers, circulators, link radios or large amounts of money. The packet-radio link can also support multiple simultaneous conversations between the ends of the path, as well as between points in the middle. Similar packet-radio "networks" exist in Florida, the Northeast and Mid Atlantic states, and in several Midwestern states.

Like the man says on TV, "And what would you expect to pay for all this? Don't answer yet, there's more . . ."

Packet allows computers to speak directly to each other in their "native tongue."

Most personal computers use the ASCII code, which has 256 separate characters. Baudot, the code most often associated with the term "RTTY," has 32 characters and a trick called "shifting," which allows an additional 26 characters to be recognized. Transmitting ASCII computer characters over Baudot RTTY causes serious problems for computer users. Packet radio can transmit ASCII characters with no restrictions or shifting.

Packet is fast.

Packet is faster than Baudot RTTY because of technical standards, equipment availability, convention and regulatory issues. For whatever reason, you won't see much Baudot above 100 words per minute (WPM). On the other hand, you won't see much packet below 360 WPM on HF or below 1440 WPM on VHF. We are starting to see packet at 11,500 WPM on 220 MHz and above, and packet has been sent experimentally at 300,000 WPM on the 70-cm band. I've taken a small liberty in expressing packet radio speeds in WPM; the actual speeds in bits per second (bit/s) are 300, 1200, 9600 and 250,000, respectively. You say that you can't type 300,000 words per minute, or even 360? Keep reading—packet isn't just for typists.

Packet provides "non-realtime" communications.

What this means is that you and the person you are talking to don't have to be home at the same time. Much of the present use of packet radio is leaving messages for others on a centrally located *bulletin board.* A bulletin board is a message-storage device, usually maintained at the home of a local ham. If the person you want to talk to isn't on the air when you are, you can leave a message for him or her on the bulletin board. The message can be about anything: plans for the breakfast meeting tomorrow, the fact that you worked the Clipper-ton DXpedition on 160 meters, your new antenna, etc. Although similar systems have been available on traditional RTTY systems, packet lends itself nicely to bulletin-board operation. Because packet is fast and many users can share the same channel, a properly designed mailbox can share the frequency with several non-mailbox conversations, or several mailboxes can be on the frequency at the same time without mutual interference.

Packet-Radio Frequencies

Packet is in use on many bands, with the most concentrated activity being on 2 meters. The national packet community is attempting to get the frequencies from 145.0 MHz to 145.1 MHz allocated to digital use, with at least the channel at 145.01 MHz available throughout North America. The following is a list of areas where official action by that area's frequency coordination body has set aside 145.01, 145.03, 145.05, 145.07 and 145.09 MHz for digital communication: Alaska, Northern California, Southern California, Connecticut, Delaware, District of Columbia, Maine, Maryland, Massachusetts, Minnesota, New Hampshire, New Jersey, New York, North Carolina, Pennsylvania, Rhode Island, South Carolina, Tennessee, Texas, Utah, Vermont, Virginia, Washington, West Virginia.

The Florida Repeater Council (FRC) has coordinated 145.01 MHz as the statewide 2-meter packet frequency. A request for 145.03 .05, .07 and .09 MHz is pending. Other states with action pending include Arizona, Illinois, Missouri, Nebraska and Oklahoma.

Many areas have activity on frequencies other than 145.01 MHz. These include Southern California, 145.36 MHz and 146.745/.145 MHz; Salt Lake City, 146.10/.70 MHz; southern and central Illinois, Iowa, east Missouri and Michigan, 147.555 MHz; southern Wisconsin and northern Illinois, 144.95 MHz.

If you are in doubt about what frequency to use, listen on 145.01 MHz.

On HF, 14.103 MHz and 10.147 MHz are the popular packet frequencies. HF packet stations transmit at 300 bauds, using a 200-Hz shift (Bell 103 standard).

Packet is information transfer.

Because of these characteristics, packet lends itself well to connecting a central store of information, usually called a *host system,* to a local network of users. For example, in Southern California we have a host run by WB6YMH that is kept stocked with the latest Amateur Radio information available. Electronic versions of the *ARRL Letter, Gateway* (the ARRL packet-radio newsletter), the *W5 YI Report, AMSAT Satellite Report* and newsletters from several other organizations are made available via packet radio and the host system to any suitably equipped amateur in the area.* A typical issue of the *ARRL Letter,* around 20,000 characters, would take about an hour to send at the standard RTTY speed of 60 WPM. At the standard VHF packet rate of 1200 bit/s, it takes about 3 minutes.

Now, what would you pay for all of this? Wait, there's more . . .

Packet is 97.1(b).

Part 97 of the FCC rules under which we live states the purpose of the Amateur Radio Service. One of the subparagraphs contains these words: "Continuation and extension of the amateur's proven ability to contribute to the advancement of the radio art." One of the better recent examples of amateurs advancing the radio art is the current activity in packet radio. Packet didn't originate in the Amateur Radio Service, but we have taken the basic idea and have shaped it into things that didn't exist before, or which have a slant different from what has been tried before. We have also added the traditional amateur touch-extremely low cost. Amateur-designed and -built packet-radio controllers flew in official weather planes through the eye of a hurricane. Army and Navy MARS stations are integrating amateur packet-radio technology into their activities. Several commercial manufacturers have taken the amateur-designed controllers and have begun to sell them both in and out of the amateur market.

Packet is satellites.

AMSAT-OSCAR 10 is an excellent medium to use for packet radio. Large amounts of data have been sent across the continent via OSCAR 10. Using a special packet device called a *teleport,* a packet station in northern Canada was connected through the satellite to a station in Los Angeles; data was then relayed through another packet controller to a station in San Diego. The UoSAT-OSCAR 11 satellite carries a packet-radio controller that can store 120,000 characters and retransmit them later to any other point on the globe. PACSAT, an amateur satellite currently under development, will use packet radio to store up to 4 million characters for relay between stations.

Packet is international.

The packet-radio protocol, AX.25, is now accepted as an international standard.

*Gateway, the ARRL packet-radio newsletter, is available from the ARRL. U.S. subscriptions are $6 for ARRL members and $9 for non-members.

Packet Operating Tips

Because it breaks' up a data stream into short sections called frames, packet looks, to the operator, like full-duplex communications+ Full duplex means both parties can speak at the same time (similar to a telephone call). This is analogous to "full-break-in." or OSK CW. Most amateurs wha have had previous exposure to . . keyboard-based communications are used to RTTY, which is half-duplex—only one side talks at a time. Most packet controllers can be adjusted to display characters from the other station only when you aren't typing yourself; this prevents your characters from being mixed on the screen with incoming characters. Sometimes, a full-duplex packet conversation can be very natural, with rapid exchanges of ideas. At other times, it can be very confusing, with the answer to one question coming in while another question is going out. Different areas of the country have came up with various ways to make a packet conversation appear more RTTY-like.

Once a "connection" is established (refer to the article text) you may begin typing. Some areas use K, BK, O or > at the end of a thought to say, "Okay, I'm done. It's your turn to transmit." Other areas simply put a blank line between thoughts to invite the other station to respond.

That's about all there is to a packet conversation. Most TNCs send a packet after you press the ENTER or CR key. It is considered polite to use this key when you reach the end of a 40- or 80-character line, rather than continuously typing and letting lines wrap around the edge of the screen.

To end a conversation, some areas use SK at the end of a line; others may add U DISC. This is short for "I'm done talking. If you're done talking, go ahead and disconnect." You can then tell your TNC to break the connection, freeing your and the other station's TNC for a new conversation.

If your TNC is so equipped, you will occasionally see a message like

CONNECT REQUEST: NK6K

This means NK6K has attempted to connect to you. This is like the "call waiting" feature on some telephones; you may disconnect from your current conversation and connect to the new station, or you may simply ignore the attempt. In any case, the other party will have received a "station busy" message, so the operator will know that you are on the air, but not available.

The best way to get started, as always, is to listen for a while, and then jump right in. It is simple to monitor the channel, or "read the mail." On the TAPR/Heath/AEA/Kantronics TNCs, simply type MON ON at the CMD: prompt. You'll see all the traffic on' the channel, with the call signs of both stations added to the beginning Of each message.

When you are ready to make a packet-radio contact, keep these suggestions in mind:

1) Transmit only short, infrequent CQ or QST packets, and don't route them through several repeaters.

2) Remember fhat you are sharing a channel; don't start long program or message transfers if there is already a lot of activity an the frequency,

3) Move to 8 frequency with no digipeaters an it whenever you can connect directly to the station that you want to work.

4) Have fun!

Amateurs worldwide are working together on future standards and applications. Papers from Japan and Germany appeared in the *Proceedings of the Fourth ARRL Amateur Radio Computer Networking Conference,* and North American papers were presented at last year's packet-radio meeting in Sweden. Three amateur satellites, JAS-1, Phase III-C and PACSAT will all use the same basic access methods. These satellites involve many parts of the world, including Canada, Germany, Japan, the U.K. and the U.S.

Packet is digital.

Experimentation with non-character communications has just started. That makes this a discussion of the future, which I said I wouldn't do till the end, but . . .

Packet is not limited to character communications. Take two SSTV units with digital outputs, plug them into packet controllers, and send absolutely error-free pictures. Better yet, store the pictures on the local host system for retrieval anytime. Digitized voice can be sent over packet radio. Several voice repeaters could share the same high-speed digital network for cross-country linking. Using packet techniques and digit al compression technology, medium-scan TV that approaches fast-scan quality can be sent at 56,000 bit/s and can be routed over high-speed packet nets with other traffic. You're putting us on, right? Nope. Check the cover; this isn't the April issue.

The Hard Part

Here's the technical part, snuck in at the middle. Don't worry, it will be over quickly. The secret of why packet can do all these amazing things is buried down in the bowels of the *protocol.* The protocol is a language spoken by the computer in your packet controller. The protocol is complex; the description of it takes many pages of nearly incomprehensible computerese, and several books have been written about it. Fortunately, you need never know how it actually works, just as you don't have to know how to alloy copper and zinc to pound brass. After all, you've just paid for a computer to understand the protocol for you. The protocol used by most amateurs is called AX.25, and probably represents the largest number of specific rules ever voluntarily agreed to by a large number of hams.

What the packet technique does is break the data sent to it into small pieces called *packets.* Several *addresses* are added to the front of the packet. An address is usually an amateur call sign. There are always at least two addresses: that of the sending station and that of the intended recipient. There may also be some addresses of stations that are supposed to repeat the packet. A Frame Check Sequence (FCS) is added to the end of the packet. The FCS is the answer to a calculation that is performed on the rest of the information in the packet. That's a packet.

The breaking up of the data into small parts allows several users to share the channel; packets from one user are interleaved with packets from another user. The address section allows each user to separate things intended for him from things intended for other users. The addresses also allow each packet to be relayed through many stations between its source and its eventual destination. The FCS allows the receiving station to make sure the data has been received correctly. The same calculation is performed on the data by the receiving station as was performed by the sending station that placed the FCS in the packet in the first place. If the FCS calculated by the receiving station matches the one sent by the transmitting station, the data is correct.

Computerized Radiograms

Traffic handlers will have recognized by now that this is the computerized equivalent of what they have been doing since the beginnings of Amateur Radio. Station of Origin, To address, a Check Number and formal procedures for relaying a message are all part of packet radio. Packet is putting the "RR" back in "ARRL."

The last piece of the pie is the acknowledgment procedure. When a packet is sent out, the sender expects an acknowledgment (ACK) that the packet was received correctly. If the ACK is not received, the packet is retransmitted. The receiver only ACKs the packet if it was received with a correct FCS. This protects a packet conversation from fades, static, collisions (when two packet stations transmit at the same time), adjacent-

channel interference and other problems common in amateur communications.

What Does It Look Like?

So far we've talked about what packet can do for you and about how it does it. But what does a packet contact look like? In the following examples, we'll look at the procedures used by the TAPR, AEA, Heath and Kantronics TNCs. Other TNCs follow similar, but not identical, procedures.

First, you must tell the TNC your call sign. For example:

MYCALL NK6K

is the command to enter a call sign. Most TNCs allow you to change your call sign at any time and have a way to remember it while the power is off.

'As in all other modes of Amateur Radio, packet allows you to "read the mail" or monitor channel activity. This is called the *monitor mode,* and looks like this:

WA6JPR> WB6YMH: HELLO SKIP, WHEN IS THE NEXT OSCAR 10 PASS?
WB6YMH >WA6JPR: HANG ON WALLY, I'LL TAKE A LOOK.

The call signs of the stations involved appear as "from> to," and the contents of the packet appear after the ":". In this manner, you can monitor all traffic on the frequency. You can also watch for a station calling CQ, which might look like this:

WB6HHV >CQ: MIKE IN SAN DIEGO LOOKING FOR ANYONE IN SIMI VALLEY.

You can send a CQ by entering the *conversation mode* of the TNC. You go to the conversation mode by typing:

CONVERSE

You can then type your CQ:

MIKE IN SAN DIEGO LOOKING FOR ANYONE IN SIMI VALLEY.

Your TNC adds your call as the FROM address, and CQ as the TO address. The receiving station's TNC adds these addresses to the front of the displayed text.

You answer a CQ or establish a contact by using the CONNECT command. This "connects" your TNC to another station and begins the acknowledgment procedure discussed earlier. An example of a connect command is

CONNECT W6IXU VIA WA6OZJ,K6TZ,WB6DAO

This asks for a connection between you and W6IXU routing through (via) three other stations. When the connection has been established, the TNC notifies you by printing

***CONNECTED TO W6IXU

This means that the computer in your TNC has exchanged some preliminary information with the other TNC and is ready to proceed. If the other station had already been in a connection with a third TNC, you would get a busy signal:

***W6IXU BUSY

If W6IXU is not on the air, your TNC would make several attempts to establish the connection and then print a message telling you that it has not succeeded.

Assuming you get connected, everything you send to your TNC will now be sent to W6IXU with all the error checking and retransmission just described. Each time you hit the ENTER or RETURN key, a packet is formed and sent. Packets received from the other station are displayed between the lines you enter, much as if a full-duplex RTTY QSO were taking place.

When you are done with the conversation, you disconnect by entering CONTROL-C and typing DISC.

The commands and scenario above are all you need to know to carry on a packet-radio QSO. There are many other options (around 60) and several other combinations of connected and monitor modes, but they are like the 40 knobs, switches and meters on most modern HF rigs; there are operators who constantly twiddle, and those who only use the push-to-talk switch or the key.

So What Are You Waiting For?

We've only touched briefly on what can be done with packet and mentioned even less the technical details of how it works. For some, packet is an end to itself— experimenting with new ways to transfer data. For others it is just a tool-anew way to pass traffic, spot tornadoes, run a parade, score points on Field Day or meet new people. To find out more, look into any of the references listed at the end of this article. Or, wait for the second part of this article, which describes a TNC in detail. See you on packet!

Bibliography

AX.25 Amateur Packet-Radio Link-Layer Protocol. Newington: ARRL, 1984.

First ARRL Amateur Radio Computer Networking Conference. Newington: ARRL, 1981.

Fourth ARRL Amateur Radio Computer Networking Conference. Newington: ARRL, 1985.

Johnson, L., "Join the Packet Radio Revolution." ***73 Magazine,*** Sept. and Oct. 1983, and Jan. 1984.

Morrison, M., and D. Morrison, "Amateur Packet Radio" ***Ham Radio,*** July and Aug. 1983.

Rinaldo, P., "ARRL Board Approves, AX.25 Packet-Radio Link-Layer Protocol, *QST,* Dec. 1984.

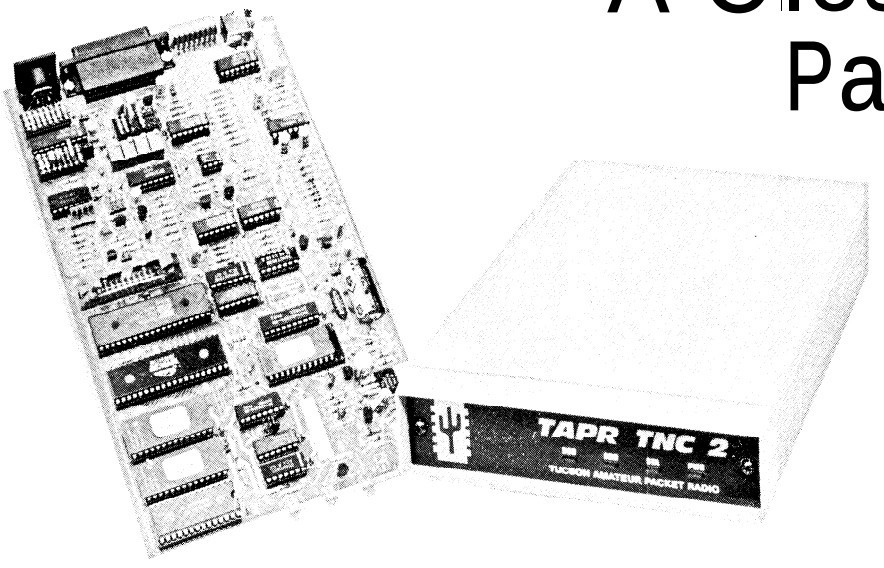
Second ARRL Amateur Radio Computer Networking Conference. Newington: ARRL, 1983.

Tannenbaum, A., ***Computer Networks.*** Englewood Cliffs, NJ: Prentice-Hail, 1981.

Third ARRL Amateur Radio Computer Networking Conference. Newington: ARRL, 1984.

The 1985 A RRL Handbook for the Radio Amateur, pp. 19-21 to 19-31. Newington: ARRL, 1984.

A Closer Look at Packet Radio



Last month we introduced you to packet radio; this month we'll look at what makes packet work-the terminal-node controller.

By Harold Price, NK6K
1211 Ford Ave., Redondo Beach, CA 90278

If you've read last month's article on packet radio, you have probably already joined the ranks of packet users and are ready to learn more about how packet works. If you haven't read last month's article, please read it now. We'll wait.

While we're waiting for them to get back, let me mention a few things that happened in the shack today, all of them fairly common occurrences on packet. You'll recall, of course, that packet is a mode of communications that allows information in digital form to be passed easily between amateur stations, that it is high speed, and that it guarantees perfect transmission of data. Packets are also easily manipulated by computer, allowing computer-assisted relaying of messages and data files.

I received a message that originated at a station in Newington, Connecticut. It had been relayed by computer on VHF to a station in Boston, where it was relayed by computer on 20 meters to a station in California. Then, I picked it up on VHF at my home in Redondo Beach in Southern California. I added some comments to the message and transmitted it on VHF through two real-time relay stations to a computer 200 miles north of me, where it will be picked up tomorrow by another station 200 miles farther to the north. I "talked" by keyboard to a station 400 miles north, in San Francisco, using four relay stations. I transmitted a copy of a packet-radio newsletter to a local ham at the speed of 120 characters per second. All of this was done using a VHF radio attached to a packet-radio controller called a *terminal-node controller* (TNC). One of the messages, by the way, discussed plans to begin transmitting data at 960 characters per second.

Is everybody back now? Good. This month, we're going to take a look at some of the technical parts of packet radio, specifically the TNC-that combination of hardware and software that does much of the hard work involved in supplying all of the services described previously.

Since it is sometimes useful to point to a concrete example of a concept under discussion, we'll use a TNC called TNC 2 (Fig. 1), designed by several hams who are part of an organization called TAPR, the Tucson Amateur Packet Radio Corporation. TAPR is a nonprofit research and development group that does work in packet radio in much the same way that AMRAD and AMSAT do work in their fields. The concepts that will be discussed here hold true for most TNCs, but are particularly applicable to the implementations by AEA, Heath and Kantronics, since their TNCs employ elements of the hardware and software previously developed by TAPR.

What A TNC Does

In professional circles, a TNC is called a packet assembler/disassembler (PAD). From this name, it is easy to figure out that a TNC's primary task is to convert data into packets, and packets into data. The TNC gets data from the user, forms it into packets and sends it out. The TNC also listens for packets, changes them back into data and passes the data to the user. There are several subsections to a TNC that allow it to do this. In the following discussion, refer to the block diagram in Fig. 2.

The User Connection

The TNC is usually attached to a local data device. This can be a terminal, a computer, a modem, a digital voice encoder or any other data-generating/using device. The TAPR TNC 2 communicates with this "user" through a serial communications port using standard RS-232-C voltage levels and signals. This means that if you have a terminal or computer that can be connected to an external modem, you can use the TNC 2. The flow of information on the user port is independent of the flow of information through the radio; the speed and data format used on the user connection don't have to match what's going on elsewhere in the TNC. Your terminal or computer doesn't even have to "know" that it is connected to a TNC. Most TNCs permit data rates between 110 bit/s and 19.2 kbit/s on the user port. While some TNCs change data rate with a software command, the TNC 2 uses switches.

The Radio Connection

The TNC must monitor the incoming signals and convert the tones it hears into ones and zeros that the rest of the TNC can deal with. It must also convert ones and zeros that the TNC wants to send into a form the radio can transmit. These jobs are performed by a demodulator and a modulator, respectively. The combination of modulator and demodulator is called a *modem.* The TNC 2 is equipped with an on-board *AFSK* modem that can be used to send data at various speeds, using various mark and space tones. On VHF packet radio, the 1200-bit/s standard is based on the Bell 202 modem standard. It uses mark and space tones of 1200 and 2200 Hz. For HF, 300 bit/s and the Bell 103 modem standard is used, using 1070- and 1270-Hz tones. As it turns out, which tone is used for mark and which for

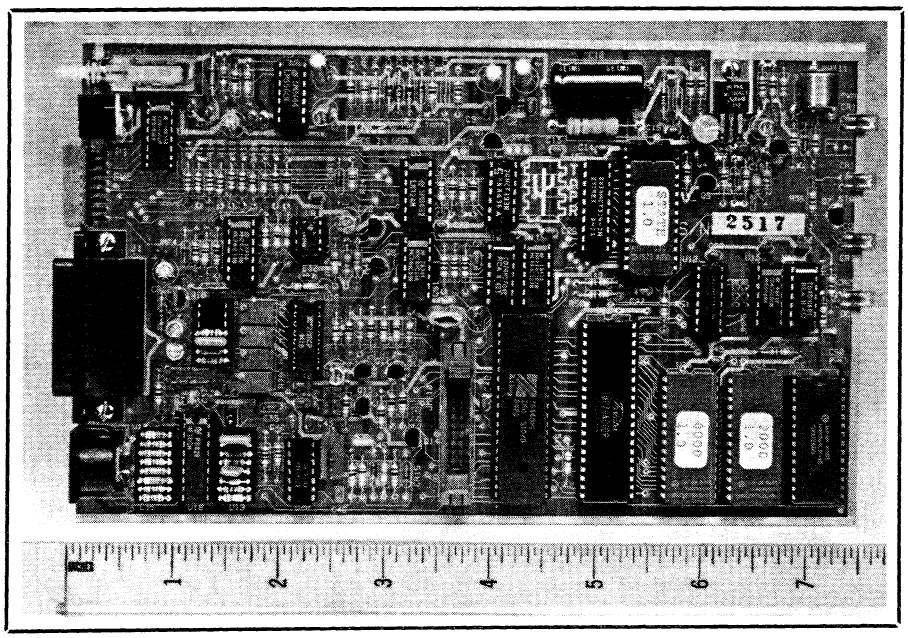


Fig. l-The PC board for the TAPR TNC 2 shows the size and parts-count reductions made possible by advances in IC technology and experience gained through design of the TAPR TNC 1.

space is of little importance on packet, since the data is sent using the special Non-Return-To-Zero-Inverted (NRZI) encoding. With NRZI, a change from one tone to the other is used to signal a zero, and no change of tone signals a 1.[1]

Although the modem in the TNC 2 is limited to 1200 bit/s or slower, TNC 2 and several other TNCs provide a way to bypass the on-board modem and use an external modem. A *modem disconnect* jack is available for this purpose, and, with the correct external modem, the TNC 2 will support data rates up to 56 kbit/s.

Before they are sent to the demodulator, received signals are conditioned by a switched-capacitor input filter. This is done because the frequency response of most VHF FM radios is somewhat less than optimal for easy decoding of a 1000-Hz shift. An entire article could be devoted to the intricacies of the modem, and, in fact, one has.[2]

In addition to modulating and demodulating, the TNC must control the push-to-talk (PTT) circuit of the radio. Since TNCs are sometimes used as unattended automatic repeaters, the FCC and common sense dictate that there be some protection against a TNC failure causing a long key-down period. The TNC 2 provides a timer on the PTT line that will turn off the transmitter if it is on for more than 15 seconds. Fifteen seconds is longer than it will take to transmit the longest possible group of contiguous packets.

In the TNC 2, as Fig. 2 shows, both the radio I/O functions and the user I/O functions are performed by a single Zilog serial I/O (SIO) chip. The SIO provides two independent serial I/O channels in one IC. This reduces parts count, power consumption and price over previous TAPR designs, while retaining the high speed made possible by having these functions performed in hardware.

Data Processing

Packet radio is easy for the operator, but it is no simple task for the TNC. The TNC must listen to both the user's data port and to the radio. It must watch the stream of packets, looking for packets addressed to it. It must acknowledge packets received correctly, complain about those received out of order and ignore those received with errors. The TNC must send out its own packets, keeping track of those that have been acknowledged and those that haven't. It must time several events: how long to wait for an acknowledgment, how long to wait for the transmitter to turn on, and others. The TNC must know who it is talking to (so it can tell other TNCs it is busy), keep track of the number of times a packet has been retransmitted and be able to relay (*digipeat*) other user's packets when

'Notes appear on page 20.

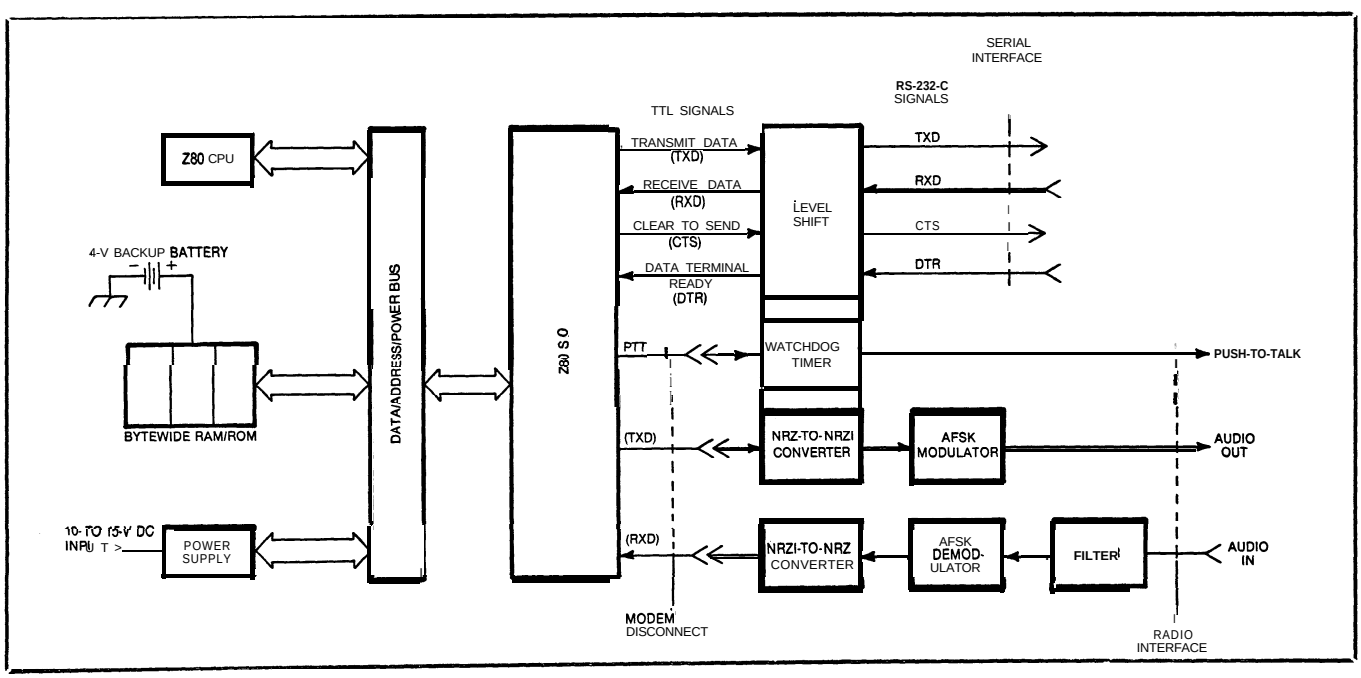


Fig. 2-A block diagram of the TAPR TNC 2. While TNC implementations vary, the services provided by the subsections in this TNC are provided in all other TNCs.

Glossary

acknowledgment (ACK)-a packet sent'& a station to confirm correct: reception of a packet or group of packets.

&dress--in amateur packet radio, a station's call sign followed by a number from 0 to 15.

AFSK—audio-frequency shift keying.

ASCII—the American National Standard Code for Information Interchange, a common binary code that specifies 7-bit representations of letters, digits, punctuation and special characters.

AX.25—the most common packet-radio protocol. AX.25 is specified in the ARRL publication *AX.25 Amateur Packet-Radio Link-Layer Protocol.* This protocol defines communication between two stations with only rudimentary relaying by intermediate stations.

bit-& single signal element, which can be either a binary 0 or 1. Groups of bits are used to convey information. Signaling speeds are usually stated in bits per second (bit/s).

bulletin board-a computer system used to store messages, files and bulletins.

connection—the imaginary "data pipeline'" that exists between twa stations that are communicating using a packet-radio protocol. During a connection, all data sent by one station will be delivered to the other station without errors.

data—bits that convey information.

digipeat—to retransmit a packet after it has been received, This is a specific function of the AX.25 protocol and is performed by *digipeaters*. With suitable radio equipment, a TNC can act as' a digipeater.

flow control—a means of controlling the rate at which data is transferred between two devices. Examples are the XON/XOFF character-based flow-control system used between the TNC and the host computer or user; hardware flow control, which uses the RS-232-C control lines; and the "sliding window" flow-control system used between TNCs.

frame check sequence (FCS)—a 16-bit number included in every packet to aid error detection. The number is the result of a calculation called a cyclic redundancy check (CRC).

gateway—A device that provides a path for data flowing between two networks,. In amateur packet radio, it may be used to connect stations on two different bands or frequencies-

host system-a computer system that can be accessed via packet radio, As well as providing the services of a bulletin board, a host system can run programs for remote users.

mail box-see *bulletin board.*

modem—the device that converts logic voltages to audio tones fur transmission, and tones to voltage levels for reception, A contraction of modulator/demodulator.

monitor mode—a mode of TNC operation during which the TNC displays all packets that it receives, not just those addressed to it,

network—a group of stations that can relay data among themselves.

network node—a station that uses a special *network protocol* (as yet, none of these exist' in Amateur Radio).

NRZI—Non-return-to-zero-inverted. A form of bit coding in which a 0 bit causes a change of state, or level, and a 1 bit causes no change In state,

packet-a group of bits that contains, along with any data being communicated, the addressing, control and error-detection information necessary for error-free communication.

protocol—a set of rules agreed upon by two stations in order to communicate.

RS-232-C—the specification of the voltage levels and signals of a serial interface.

teleport—a station that provides a link between a terrestrial network and a satellite.

terminal-node controller (TNC)—a dedicated, microprocessor-based device that communicates with other TNCs using a packet protocol and with the user's terminal or computer using serial ASCII (RS-232-C).

user interface—the set of commands that a user can enter into the TNC and the messages or responses returned by the TNC to the user.

called upon to do so. It must listen to the radio and not transmit if another signal is on the air.

It takes a computer to keep track of all this. The last item in the preceding paragraph has proven especially difficult for human operators to do; just listen to 20 meters on any contest weekend. The processing power for the TNC 2 is provided by a Zilog Z80® CPU, running at 2.45 MHz. The TNC 2 has three memory sockets. These sockets usually hold 16 kbytes of read-only memory (EPROM) for program storage and 8 kbytes of read/write memory (RAM) for data storage. Using the largest available ICs, it is possible to get a total of 56 kbytes of memory into the three sockets, which leaves some room for expansion of the TNC software.

A lithium battery supplies backup power to the TNC 2 RAM when main power is removed from the TNC. This is exactly the same scheme that is used in most mobile or hand-held VHF/UHF radios; a small internal battery keeps memory active, so the rig can "remember" repeater frequencies and offsets. TNC 2 uses its battery-backed-up RAM to remember your call sign and the settings of the more than 70 variable parameters used to configure the TNC to your needs.

Software

Writing the software used to control a TNC is one of the more difficult programming tasks required in Amateur Radio. It is rivaled only by the software in OSCARs 10 and 11, and the software on some of the more complex hilltop repeater and remote-base systems.

A TNC requires two different types of software. First, the TNC is a computer speaking to other computers. It does this using an agreed-upon language called a *protocol.* A good protocol definition is very precise, leaving no room for interpretation or surprises.. The packet-radio protocol in widest use today is called *AX.25*SM pend.. The specification of AX.25, 40 pages of protocolese,, is perhaps the most comprehensive set of rules that any large segment of the Amateur Radio population has ever agreed to live by (apart from Part 97 itself).[3] AX.25, however, is only the first floor of a multiple-story house that is being built by the packet-radio community. The task of specifying (and agreeing upon!) protocols has really just begun. Although the AX.25 protocol is sometimes hard for humans to understand, it is just the type of well-defined task at which computers excel.

The second type of software in a TNC is the *user interface* program. Here things are a little less certain; humans have an amazing propensity for attempting things never before tested, tried, planned or even imagined. What humans lack in speed, they make up for with the talent for unerringly choosing the wrong thing to do at the wrong time. There are no reliable methods for guessing what people will do next:. Computers and computer programmers do not like this kind of behavior. Thus, the amount of software written to talk to the user usually exceeds the amount written to talk to the other TNCs.

Writing TNC code is not for the faint of heart, but it can be a rewarding experience. TNC 2, like most other TNCs, comes with all the necessary protocol and user software stored in EPROM. This means that if software updates are necessary, they can be accomplished by merely changing a memory IC or two.

Power Supply

TNC 2 requires an external 10- to 15-V dc supply. An on-board switching-mode power supply converts that input to regulated +5 V, and -5 V. The supply also provides -7 V for the RS-232-C outputs. (The two RS-232-C output levels

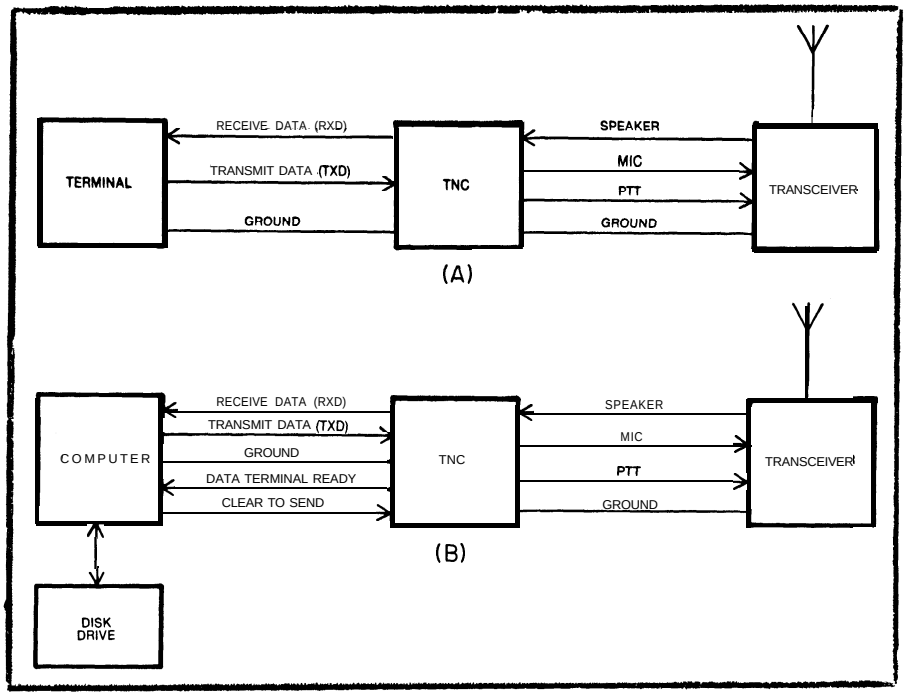


fig. 3-A shows the simplest TNC, radio and terminal connections. In B, the additional hardware flow control lines between the computer and the TNC make it possible to transmit large data or message files.

are -7 V and the positive input voltage.) Populated with NMOS ICs, the TNC 2 draws 260 mA at 12 V. For lower-power consumption, the TNC 2 was designed so that it could be built with CMOS parts. With CMOS ICs, the TNC draws less than 120 mA at 12 V.

Status Indicators

You never quite feel you've gotten your money's worth unless you've got flashing lights. TNC 2 has four of them: POWER ON (the function should be obvious); CONNECTED, lights when a connection to another TNC has been established; DATA CARRIER DETECT glows when a mark or space tone is heard; and STATUS is lit when the TNC has sent a packet but has not yet received an acknowledgment for it.

Hooking It Up

The TAPR TNCs grew from a single overriding desire: to make packet radio as easy as possible to use! The Bell 202 modem standard and 1200-bit/s data rate are used because this is as fast a signal as can be easily forced into the microphone jack and taken from the speaker terminals on most VHF/UHF radios. To go faster requires a direct connection to the modulator and discriminator. This is not particularly difficult, but would limit packet operation to those with older, larger rigs or to surgeons and jewelers with the steadiness of hand, keenness of eye and tiny tools necessary to make modifications to shirt-pocket radios.

Since the TNC was designed for easy installation, it should not be surprising that hooking up a TNC to your station is very simple. As shown in Fig. 3A, you can get by with three wires to your computer or terminal and four to your radio. The hard part will be finding a proper mic connector for your radio! Just remember, when connecting a TNC to your radio, think of the TNC as a microphone and a speaker; when connecting a TNC to a computer, think of the TNC as a modem.

If you use your TNC only for RTTY-like typing contacts, you need to connect only three wires between the TNC and your computer: one for data from the computer to the TNC (TXD), one for data from the TNC to the computer (RXD) and one ground (Fig. 3A). If you want to use your computer to send large files or messages to your TNC, then you must provide a way for the TNC to control the stream of data coming from your computer. This is called *flow* control. Flow control is required because your computer can send data to the TNC faster than the TNC can send data to the receiving station. If you send a stream of data at 1200 bit/s to your TNC, retries-caused by collisions, dropped packets or other mishaps-will cause the limited RAM in your TNC to fill up with characters waiting to be transmitted. Your computer must be prepared to wait when the TNC memory gets full. Two flow-control methods are available on the TNC 2: hardware flow control, using the CTS and DTR lines on the serial port (Fig. 3B), or software flow control, using the ASCII XON and XOFF characters.

As mentioned earlier, on TNCs that have a modem disconnect, you are not limited to use of the on-board modem. The TNC 2 is capable of running at 56 kbit/s with an appropriate external modem. A 9600-bit/s modem that connects to the modem disconnect has been designed, and other, even faster, modems are under discussion.[4] High speeds will be used primarily for communication between gateways and network nodes, but that's a topic for another day.

Wrapping It Up

We've seen what a TNC is and what it does. We've used the TAPR TNC 2 as a specific example. I'd like to mention the chief architects of that project. Paul Newland, AD7I, did the hardware design, and Howard Goldstein, N2WX, did the software. Steve Goode, K9NG, provided input on the modem, and other design input and review came from Pete Eaton, WB9FLW, and Lyle Johnson, WA7GXD.

What happens next? I'd like to suggest that you stop reading about packet radio and do something about it. Packet radio is a young enough part of our hobby that you can get in on the ground floor and have a very real effect on the future growth and direction of computer networking in the Amateur Radio Service. You might just also affect the future of Amateur Radio itself.

If you'd like to see more *QST* articles about packet, including things on high-speed modems, gateways, n-port digipeaters and network access ports, send a letter to the editors. If they hear that there is interest in packet radio, they are more likely to publish packet-related articles. To stay current in the meantime, you can join any of the several packet radio clubs that print newsletters. Also, the biweekly ARRL packet radio newsletter, Gateway, provides short reviews and summaries of packet development activity.

See you on packet!

Notes

[1] ***The 1985 ARRL Handbook for the Radio Amateur,*** p. 19-25.

[2] M. Morrison and D. Morrison, "Designing the TAPR TNC Audio Input Filter," ***Proceedings of the 2nd ARRL Amateur Radio Computer Networking Conference.*** Available from ARRL for $9.

[3] ***AX.25 Amateur Packet-Radio Link-Layer Protocol, Version 2.0, Oct.*** 7984. Available from ARRL for $8.

[4] S. Goode, "Modifying the Hamtronics FM-5 for 9600-Baud Operation," ***Proceedings of the 4th ARRL Amateur Radio Computer Networking Conference,*** available from ARRL for $10.

I would like to get in touch with. . .

❑ anyone using an EAGLE IIE computer to work RTTY, AMTOR and packet. Jack Clark, W9HJM, 93 Downing Dr., Chatham, IL 62629.

❑ anyone with a four-section electrolytic can capacitor rated at 20 μF/ 20 μF/ 20 μF/ 30 μF at 650 V. Charles Schramm, Jr., KA2JLC, 28-28 35 St., Long Island City, NY 11103.